Multi-limited Lindenmayer Systems

Von der Carl-Friedrich-Gauß-Fakultät
Technische Universität Carolo-Wilhelmina zu Braunschweig

zur Erlangung des Grades

Doktor-Ingenieur (Dr.-Ing.)

genehmigte Dissertation

von	Dipl.-Math. Markus Seemann
geboren am	27.04.1968
in	Wolfsburg

Eingereicht am:	18.07.2007
Mündliche Prüfung am:	22.11.2007
Referent:	Prof. Dr. Dietmar Wätjen
	Institut für Theoretische Informatik
	TU Braunschweig
Korreferent:	Prof. Dr. Jürgen Dassow
	Fakultät für Informatik
	Universität Magdeburg

(2007)

Preface

My thanks go to my mentor Prof. Dr. D. Wätjen for his scientific support and productive discussions as well as for accepting the task of compiling the report of this thesis. I also thank Prof. Dr. J. Dassow for accepting the task of issuing the co-report of this thesis.

I thank Mrs. S. Jörger and Mr. S. Griebel for their detailed review of this work. Finally, I thank my parents and all my friends for their sympathy.

Braunschweig, December 2007

Kurzfassung

Der Biologe und Mathematiker Aristide Lindenmayer begründete die Theorie der L-Systeme. Das ursprüngliche Ziel dieser Theorie ist die Bereitstellung mathematischer Modelle zur Untersuchung des simultanen Zellwachstums fadenartiger Organismen. Da L-Systeme als eine Art von Ersetzungssystemen definiert sind, sind ihre erzeugten Sprachen, d.h. die Mengen der durch Zeichenketten beschriebenen Organismen, ebenfalls Gegenstand der Theorie der formalen Sprachen. Diese Theorie klassifiziert formale Sprachen sowie ihre Erzeugungsmechanismen gemäß ihrer Eigenschaften, wie z.B. Erzeugungsmächtigkeit oder Entscheidbarkeit.

Als ein Sprachen-erzeugender Mechanismus, der zwischen der rein sequentiellen Ersetzung kontextfreier Grammatiken und der rein parallelen Ersetzung von L-Systemen liegt, sind k-limitierte L-Systeme von D. Wätjen eingeführt und untersucht worden. In der Biologie können diese Systeme als Organismen interpretiert werden, deren simultanes Zellwachstum beschränkt ist durch individuelle Nahrungsvorräte mit einer einheitlichen endlichen Kapazität k.

Die in dieser Arbeit betrachteten mehrfach-limitierten L-Systeme bilden eine Verallgemeinerung der k-limitierten L-Systeme, indem sie für jeden Zelltyp a einen individuellen Nahrungsvorrat mit einer spezifischen Kapazität $\kappa(a)$ anstelle der einheitlichen Kapazität k vorsehen.

Diese Arbeit führt mehrfach-limitierte L-Systeme ein und definiert eine geeignete Kategorisierung der von ihnen erzeugten Sprachfamilien anhand der verwendeten bzw. erlaubten Mengen von Limits $\kappa(a)$. Zunächst wird ein intuitiver Zugang zu den verschiedenen Mechanismen der L-System-Varianten ermöglicht, indem eine Methode zur grafischen Interpretation von L-Systemen, die sogenannte Turtle-Interpretation, vorgestellt wird. Hierzu werden geeignete Computer-Programme für einen Turtle-Interpreter sowie für frei programmierbare Simulatoren von mehrfach-

limitierten, k-limitierten sowie uniform k-limitierten L-Systemen erstellt und ihr Quell-Code zur Verfügung gestellt.

Im Anschluss an diesen Umriss eines Anwendungsgebietes von L-Systemen außerhalb der Theorie der formalen Sprachen werden mehrfach-limitierte L-Systeme unter formalsprachlichen Aspekten untersucht. Ihre erzeugten Sprachfamilien werden bzgl. ihrer Inklusionseigenschaften untereinander, mit Wätjens k-limitierten Sprachfamilien, mit den nicht-limitierten Sprachfamilien sowie mit der Chomsky Hierarchie verglichen. Die Erzeugungsmächtigkeit von mehrfach-limitierten L-Systemen wird asymptotisch verglichen mit den jeweils unterliegenden nicht-limitierten L-Systemen. Des weiteren werden die Abschlusseigenschaften der mehrfach-limitierten L-Systeme untersucht.

Ein Ergebnis dieser Untersuchung ist, dass verschiedene Mengen erlaubter Limits stets zu unterschiedlichen Sprachfamilien führen, wenn die sogenannten nicht-erweiterten mehrfach-limitierten L-Systeme betrachtet werden. Für gewisse Typen dieser Systeme charakterisiert die Mengen erlaubter Limits sogar die zugehörigen Sprachfamilien, so dass diese Sprachfamilien strikte Hierarchien bzgl. der Inklusion bilden.

Für die sogenannten erweiterten mehrfach-limitierten L-Systeme wird eine Normalform angegeben. Des weiteren wird, mit Ausnahme der propagierenden Systeme, die Erzeugungsmächtigkeit dieser Systeme nicht erweitert, wenn die Mengen erlaubter Limits durch eine beliebige Menge ersetzt wird, die aus endlichen Summen der erlaubten Limits besteht. Als eine Konsequenz hieraus ist die Familie der von den erweiterten mehrfach-limitierten L-Systemen erzeugten Sprachen gleich der Familie der von den erweiterten deterministischen k-limitierten L-Systemen erzeugten Sprachen.

Die Erzeugungsmächtigkeit von deterministischen mehrfach-limitierten L-Systemen mit nur einer Tafel sowie von propagierenden mehrfach-limitierten L-Systemen konvergiert stets gegen die Erzeugungsmächtigkeit des unterliegenden L-Systems, wenn das kleinste erlaubte Limit gegen unendlich strebt. Für die übrigen Fälle werden Gegenbeispiele angegeben.

Abstract

The theory of L systems originated with the biologist and mathematician Aristide Lindenmayer. His original goal was to provide mathematical models for the simultaneous development of cells in filamentous organisms. Since L systems may be viewed as rewriting systems, their generated languages, i.e., sets of organisms encoded by strings, are also subject to formal language theory, which aims to classify formal languages as well as their generating mechanisms according to various properties, such as generative power, decidability, etc.

D. Wätjen introduced and studied k-limited L systems in order to combine the purely sequential mode of rewriting and the purely parallel mode of rewriting in context-free grammars, respectively, L systems. In biology, these systems may be interpreted as organisms, for which the simultaneous growth of cells is restricted by the supply of some resources of food being limited by some finite value k.

In this thesis the constraint of a common limit k is relaxed in favor of individual resource limits $\kappa(a)$ for every cell-type a, which yields the new notion of *multi-limited L system*. The language families generated by such systems are then classified according to their sets of limits $\kappa(a)$.

At first, an intuitive approach to the different mechanisms of the L system variants is provided by presenting a method for the graphical interpretation of L systems, the so-called *turtle interpretation*. Suitable computer programs implementing a turtle interpreter as well as free-programmable simulators for multi-limited, k-limited, and uniformly k-limited L systems, are developed and their source-code is appended.

After this outline of an application area of L systems outside formal language theory, multi-limited L systems are investigated under aspects of formal language theory. Their generated language families are compared to each other, to Wätjen's k-limited as well as to non-limited language families, and to the families of the Chomsky Hier-

archy. Besides asymptotically comparing the generative power of multi-limited L systems to that of the underlying non-limited L systems, also their closure properties are investigated.

It arises that different sets of limits always result in different families generated by so-called non-extended multi-limited L systems. For certain subtypes of these one even obtains a characterization of the induced language families in terms of the limit sets, which results in a strict hierarchy with respect to set-inclusion.

For the so-called extended multi-limited L systems, a normal form is presented. Furthermore, except for propagating systems, their generative power does not grow if the set of permitted limits is replaced by an arbitrary set of finite sums of these limits. As a consequence of this, the family of languages generated by extended multi-limited L systems coincide with the family of languages generated by extended deterministic k-limited L systems.

Finally, two classes of multi-limited L systems are identified, for which the generative power always converges to that of the underlying non-limited L system, as the minimum of the limit set approaches infinity: deterministic multi-limited L systems with just one table and propagating multi-limited L systems. For other cases counter examples are presented.

Contents

1 Introduction **1**

1.1 Motivation 1

1.2 Goals of this Thesis 5

2 Basic Definitions and Facts **7**

2.1 Basics of Formal Language Theory 7

2.2 Lindenmayer Systems and Languages 15

2.3 k-limited Lindenmayer Systems and Languages 19

2.4 Multi-limited Lindenmayer Systems and Languages 23

3 Graphical Modeling using L Systems **29**

3.1 Turtle Interpretation of Strings 30

3.2 Bracketed 0L Systems 34

4 Multi-limited T0L Systems and Languages **37**

4.1 Inclusion Relations 37

4.1.1 Comparison amongst the Families of mlT0L Languages 37

4.1.2 Comparison with the Families of T0L Languages 49

4.1.3 Comparison with the Chomsky Hierarchy 50

4.2 Closure Properties 52

5 Multi-limited ET0L Systems and Languages 57

5.1 Normal Form 57

5.2 Inclusion Relations 60

5.2.1 Comparison amongst the Families of mlET0L Languages . . . 61

5.2.2 Comparison with the Families of ET0L Languages 80

5.2.3 Comparison with the Chomsky Hierarchy 81

5.2.4 Decidability Results 83

5.3 Closure Properties 85

6 Asymptotic Inclusion Relations 91

6.1 General Properties of mlET0L Systems 91

6.2 Converging sequences 93

6.2.1 A further result concerning ED0L systems 97

6.3 Non-converging sequences 99

7 Summary 103

7.1 Summary and Open Problems 103

7.2 Suggestions for further Research 105

A Source Code 107

A.1 Examples of Turtle Interpretations 107

A.1.1 Tree-like Branching Structures 107

A.1.2 Hexagonal Gosper Curves 108

A.1.3 Combination of islands and lakes 109

A.2 Bracketed Turtle Interpreter 109

A.3 Simulator for ml0L systems 121

A.4 Simulator for kl0L systems 132

A.5 Simulator for uniformly kl0L systems 144

A.6 Auxiliary Class "DListTemp.h" 155

A.7 Auxiliary Class "mathe.h" 159

Bibliography **161**

Chapter 1

Introduction

1.1 Motivation

In this thesis multi-limited Lindenmayer systems and languages are introduced and investigated with respect to formal language theory. In order to define the goals of this work, a survey of the context of multi-limited Lindenmayer systems and languages regarding formal language theory is given first.

In 1956, formal language theory was introduced by N. Chomsky within the scope of his mathematical investigations of natural languages in [2]. He described formal languages by special types of so-called rewriting systems. Rewriting systems were defined by A. Thue in [35] in 1914 as follows: Consider a finite set Σ of symbols, the so-called alphabet. A rewriting system step by step derives strings or words over the alphabet Σ from an initial word ω by replacing substrings in ω iteratively according to particular replacing rules. These replacing rules were called *productions* by Thue.

Chomsky named his rewriting systems grammars. Typically, they replace exactly one string per step. As an additional control mechanism, Chomsky partitioned the alphabet Σ into two disjoint alphabets Δ and $\Sigma \setminus \Delta$, the set of terminal respectively non-terminal symbols. He defined that all those terminal words over Δ which can be generated by a grammar G from an initial symbol $S \in \Sigma$ constitute the formal language $L(G)$. Chomsky classified his grammars by four different types according to certain formal properties of their productions. The families of languages generated by these types of grammars Chomsky called the family of regular, context-free, context-sensitive, and recursively enumerable languages (see [3]). Together with the family

of all finite languages they form a hierarchy with respect to strict inclusion, the so-called Chomsky Hierarchy, as indicated in Figure 1.1. The Chomsky Hierarchy is used in formal language theory as a standard of comparison regarding the generative power of formal language families.

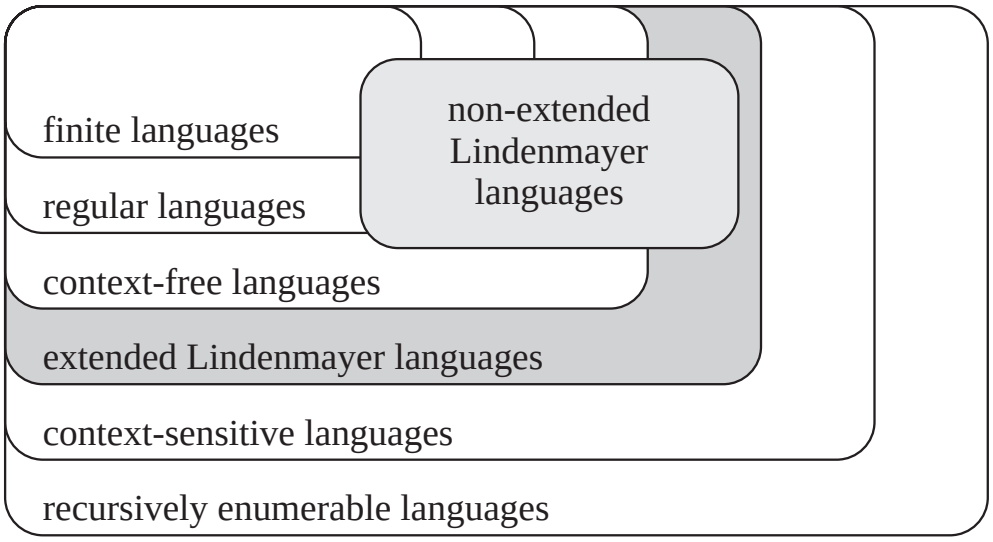

Figure 1.1. Venn diagram of the families of Lindenmayer languages and the families of the Chomsky Hierarchy.

Since its introduction, formal language theory has been an interdisciplinary area of research because in various scientific disciplines, a formal description of specific languages is needed. For example, context-free grammars, like the Backus Naur Form (BNF), which sequentially rewrite only one single non-terminal symbol at a time, are today used to specify the syntax of programming languages or to construct compilers. Thus, formal language theory often received impulses from outside.

This was also the case when the biologist and mathematician Aristide Lindenmayer originated the theory of Lindenmayer systems, abbreviated as L systems, in 1968 (see [16, 17]). The original aim of this theory is to provide mathematical models for the simultaneous development of cells of filamentous organisms. In a first approach, Lindenmayer defined L systems as linear arrays of finite automata. Later he redefined them by more suitable constructs which are similar to Chomsky's context-free grammars, but which do not necessarily use non-terminal symbols. In case they use non-terminal symbols, they are called extended L systems. Otherwise, they are called non-extended L systems. In contrast to the sequential rewriting of Chomsky's

context-free grammars, L systems always apply their context-free rewriting rules to all letters of a word in process parallely. The languages generated by L systems were investigated in [15, 18, 24, 26, 31]. It turned out that both the family of extended L systems and the family of non-extended L systems, respectively generate new families of languages that do not coincide with any of the Chomsky families, as shown in Figure 1.1.

Further language generating mechanisms combining both the pure sequential rewriting of context-free grammars and the pure parallel rewriting of L systems, also were subject to research. For example, k-limited L systems and uniformly k-limited L systems were introduced and investigated by D. Wätjen in 1988 (see [36]) and 1990 (see [38]), respectively. A k-limited L system processes a given word w by simultaneously rewriting all, but at most k occurrences of each symbol in w. A uniformly k-limited L system behaves similarly, but does not distinguish between different symbols when it determines the number of symbols to be rewritten. This means that a uniformly k-limited L system processes a given word w by simultaneously rewriting all occurrences of each symbol in w, but at most k in total.

In biology, these systems can be interpreted as organisms whose simultaneous cell growth is restricted by some resources of food each having the same capacity k. In the k-limited case, each cell-type has its own resource. In the uniformly k-limited case, there is only one common resource for all cell-types. Due to his investigations Wätjen proved that the families of formal languages generated by k-limited and uniformly k-limited L systems are also new (see [6, 32, 33], [36]–[46]).

Some differences between non-limited, k-limited, and uniformly k-limited L systems are visualized in Figure 1.2.

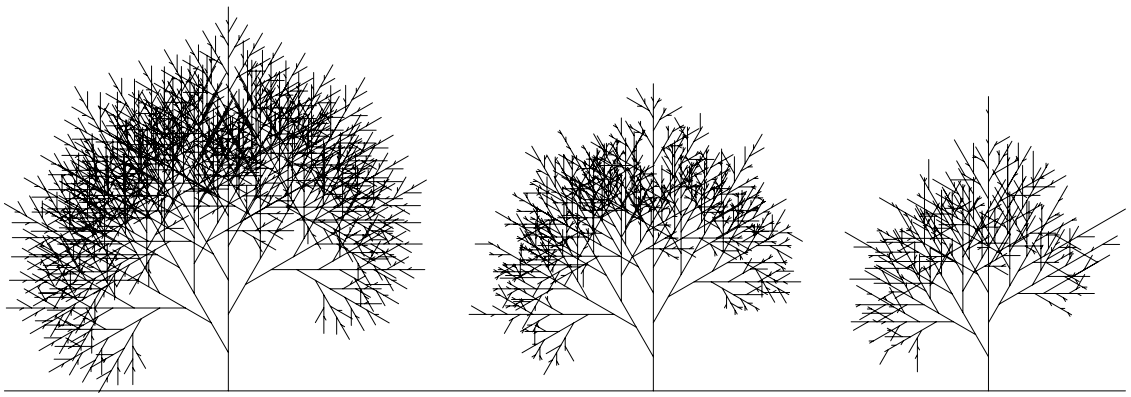

Figure 1.2. Visualization of three words generated by similar non-limited, k-limited, and uniformly k-limited L systems, respectively (from the left to the right; see also Appendix A.1.1).

Each of the three trees represents an organism which was generated by the respective type of L system, but from the same initial organism, by using the same rewriting rules, and within the same number of derivation steps. The left-most tree was generated by the non-limited L system and looks very regular which seems rather unlikely to be found in nature. The right-most tree was generated by the uniformly k-limited L system. The effect of the limitation is obvious: the most frequent cell-types have the highest probability to be rewritten. In this example, especially the cells of straightforward branches grow super-proportional. The middle tree was generated by the non-uniformly k-limited L system. After an initial stage of exponential cell growth, all cell-types of the organism are growing linearly as in the presented stage. This behavior seems to be more close to nature than those of non-limited or uniformly k-limited L systems.

It is interesting to note that the graphics of Figure 1.2 themselves were generated by appropriate L systems combined with a special method of graphical interpretation of strings that is called turtle graphic and is described in Chapter 3.

As a generalization of both k-limited and uniformly k-limited L systems, S. Gärtner introduced and investigated in his thesis in 1995 partition-limited L systems (see [12]). In these systems, the alphabet Σ of cell-types is partitioned into non-empty sets where all cell-types of a set are sharing one common resource of constant capacity k. In case that all these sets are singleton sets, the system behaves as a k-limited system. In case that the partition consists of only one set, a uniformly k-limited system is simulated.

For further research, Gärtner suggested a special kind of generalization of non-uniformly k-limited L systems. These so-called multi-limited Lindenmayer systems provide an individual capacity k_a instead of a common limit k for every resource of a fixed cell-type a. The tree shown in Figure 1.3 represents an organism which was generated by a multi-limited L system which started from the same initial organism and used the same rewriting rules as those examples of Figure 1.2. Yet in this example, the resource for the left-branching cell-type was ten times smaller than the resource of every other cell-type. As a consequence of this, the branches of the organism turn clockwise.

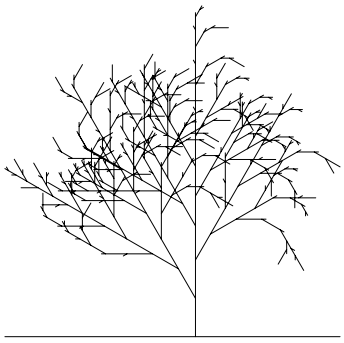

Figure 1.3. Visualization of a word generated by a multi-limited Lindenmayer system (see also Appendix A.1.1).

1.2 Goals of this Thesis

Based on the language theoretical background described so far, the goals of this thesis are defined as follows:

(1) Introduction of multi-limited Lindenmayer systems and languages:
Chapter 2 motivates how to classify multi-limited Lindenmayer systems and languages regarding families of systems and families of languages and reports of the underlying definitions and facts of formal language theory. This constitutes the basis for the subsequent chapters.

(2) Presentation of a method for the graphical interpretation of L systems:
Chapter 3 presents a method for graphical interpretation of arbitrary limited and non-limited L systems as depicted in figures 1.2 and 1.3. This provides an intuitive approach to the different mechanisms of the L system variants consid-

ered in this work. Simultaneously, it outlines an application area of L systems outside formal language theory.

(3) Investigation of multi-limited L systems under aspects of formal language theory:
Chapter 4 considers non-extended systems, i.e. systems without a non-terminal alphabet. Chapter 5 deals with extended multi-limited L systems which generalize the non-extended systems by using an additional, non-terminal alphabet. The language families generated by these systems are compared to each other, to Wätjen's k-limited language families, to non-limited language families, and to the Chomsky Hierarchy regarding inclusion. Furthermore, closure properties are investigated.

(4) Investigation of the influence of the limitation of multi-limited L systems regarding their generative power:
In Chapter 6, the generative power of multi-limited L systems is compared asymptotically to the underlying non-limited L systems for the case that the minimum capacity of the multi-limited L systems tends to infinity.

(5) Suggestions for further research:
Chapter 7 summarizes the new results of this thesis and provides several suggestions for further research.

Chapter 2

Basic Definitions and Facts

This chapter introduces multi-limited Lindenmayer systems and languages in Section 2.4. Preceding, the necessary background of formal language theory and Lindenmayer systems is represented in three steps:

2.1 Basics of Formal Language Theory
2.2 Lindenmayer Systems and Languages
2.3 k-limited Lindenmayer Systems and Languages

For further reading regarding these three topics see [29], [26], and [36] , respectively.

In this document the letter $\mathbb{Z}$ denotes the set of all integers, $\mathbb{N}$ denotes the set of all positive integers, and $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$. Furthermore, the set of all subsets of a set S is written as $\wp(S)$.

2.1 Basics of Formal Language Theory

Definition 2.1 A finite and non-empty set Σ is called an *alphabet*. Its elements are called *symbols* or *letters*. The number of symbols of Σ, which is called the *size* of Σ, is denoted by $|\Sigma|$. A *word* w *over* Σ is a finite string consisting of zero or more *occurrences* of symbols of Σ. A substring v of w is called a *subword* of w. If a subword v contains the beginning respectively the ending of w, then v is called a *prefix* respectively a *suffix* of w. The number of occurrences of a symbol a in w is denoted by $\#_a w$. For a set Δ the total number of occurrences of symbols of Δ in w

is written as

$$\#_\Delta w = \sum_{a \in \Delta} \#_a w.$$

The number $\#_\Sigma w$ is called the *length* of w and is abbreviated by $|w|$. The word of length zero is called the *empty word* and is denoted by ε. Note that ε is a word over each arbitrary alphabet. The set of all words over Σ is written as Σ^*, and $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$ denotes the set of all non-empty words over Σ. Note that Σ^* and Σ^+ always are infinite sets. Every set $L \subseteq \Sigma^*$ of words over Σ is called a *formal language (over Σ)* or just a *language (over Σ)*. L is called *finite* if L consists of finitely many words. L is called *ε-free* if $\varepsilon \notin L$. ■

Definition 2.2 The *concatenation of words* over Σ is the binary operation $\cdot : \Sigma^* \times \Sigma^* \to \Sigma^*$ where $u \cdot v = u_1 \dots u_m v_1 \dots v_n$ for arbitrary $u = u_1 \dots u_m$, $v = v_1 \dots v_n \in \Sigma^*$ with $u_1, \dots, u_m, v_1, \dots, v_n \in \Sigma$ and $m, n \in \mathbb{N}_0$. In algebra, a binary operation $\circ$ on a set M together with a special (neutral) element $e \in M$ constitutes a so-called *monoid* $\langle M, e, \circ \rangle$ if

(m1)	$\circ$ is *associative*:	$(u \circ v) \circ w = u \circ (v \circ w)$	for all $u, v, w \in M$,
(m2)	e is the *neutral element*:	$e \circ w = w = w \circ e$	for all $w \in M$

(see [20]). Obviously, concatenation of words is associative with the empty word ε as its neutral element. Consequently, $\langle \Sigma^*, \varepsilon, \cdot \rangle$ is a monoid for every alphabet Σ, the so-called *free monoid* over Σ, and brackets can be omitted whenever more than two words are concatenated. Now, the *i-th power of a word $w \in \Sigma^*$* can be defined as $w^i = w \cdot w^{i-1}$ for every $i \in \mathbb{N}$ and $w^0 = \varepsilon$. Analogously, the *i-th power of a language* $L \subseteq \Sigma^*$ is defined as $L^i = L \cdot L^{i-1}$ for $i \in \mathbb{N}$ and $L^0 = \{\varepsilon\}$ where the binary operation $\cdot : \wp(\Sigma^*) \times \wp(\Sigma^*) \to \wp(\Sigma^*)$ with

$$L_1 \cdot L_2 = \{w_1 \cdot w_2 \mid w_1 \in L_1 \wedge w_2 \in L_2\}$$

is the *concatenation of languages* over Σ. This concatenation is also associative with $\{\varepsilon\}$ as its neutral element such that $\langle \wp(\Sigma^*), \{\varepsilon\}, \cdot \rangle$ is also a monoid. In both types of concatenation the dot signs are usually omitted such that $w_1 \cdot w_2$ and $L_1 \cdot L_2$ are respectively written as $w_1 w_2$ and $L_1 L_2$ for words w_1, w_2 and languages L_1, L_2. Now, the previous definition of Σ^* and Σ^+ for alphabets Σ can be generalized to L^* and L^+ for languages $L \subseteq \Sigma^*$ by

$$L^* = \bigcup_{i=0}^{\infty} L^i \qquad \text{and} \qquad L^+ = \bigcup_{i=1}^{\infty} L^i.$$

The union L^* of all nonnegative powers of L is called *iteration* or *Kleene star* of L, and the union L^+ of all positive powers of L is called *ε-free iteration* or *Kleene plus* of L. Thus, $L^* = L^+ \cup \{\varepsilon\}$. ■

Note that always $\varepsilon \in L^*$. But $\varepsilon \in L^+$ if and only if $\varepsilon \in L$.

As mentioned above, the monoid $\langle \Sigma^*, \varepsilon, \cdot \rangle$ of words over Σ is called the *free monoid* over Σ which means that for every monoid $\langle M, e, \circ \rangle$ and every function $f : \Sigma \to M$ there exists exactly one so-called *monoid homomorphism* $h : \Sigma^* \to M$ with $h(a) = f(a)$ for all $a \in \Sigma$ (see [20]).

Definition 2.3 Given two monoids $\mathcal{M} = \langle M, e, \circ \rangle$ and $\mathcal{M}' = \langle M', e', \circ' \rangle$, a *monoid homomorphism* h is defined as a function $h : M \to M'$ preserving the monoid structure, i.e.

$$\begin{array}{ll} \text{(h1)} & h(e) = e', \\ \text{(h2)} & h(u \circ v) = h(u) \circ' h(v) \quad \text{for all } u, v \in M. \blacksquare \end{array}$$

Two special kinds of monoid homomorphisms are playing an important role in language theory and especially in the theory of L systems.

Definition 2.4 Given two alphabets Σ_1 and Σ_2, every monoid homomorphism $\sigma : \Sigma_1^* \to \wp(\Sigma_2^*)$ is called a *substitution* and every monoid homomorphism $h : \Sigma_1^* \to \Sigma_2^*$ is simply called a *homomorphism*. As mentioned above, it suffices to define h and σ only on Σ_1 to obtain the whole monoid homomorphism on Σ_1^*, respectively. Homomorphism h and substitution σ are called *ε-free* if $\varepsilon \neq h(a)$ respectively $\varepsilon \notin \sigma(a)$ for all letters $a \in \Sigma_1$. Substitution σ is called *finite* respectively *non-empty* if every language $\sigma(a)$ is finite respectively non-empty for all $a \in \Sigma_1$. Obviously, by identifying elements with their singleton sets, every finite substitution σ with $|\sigma(a)| = 1$ for all $a \in \Sigma_1$ can be interpreted as a homomorphism and vice versa.

Homomorphism h and substitution σ both can be extended to monoid homomorphisms which map subsets of Σ_1^* to subsets of Σ_2^* by setting

$$h(L) = \bigcup_{w \in L} \{h(w)\} \qquad \text{and} \qquad \sigma(L) = \bigcup_{w \in L} \sigma(w)$$

for every language $L \subseteq \Sigma_1^*$. The *inverse homomorphism* h^{-1} is defined as usual, i.e.

$$h^{-1}(w_2) = \{w_1 \in \Sigma_1^* \mid h(w_1) = w_2\} \qquad \text{and} \qquad h^{-1}(L) = \bigcup_{w \in L} \{h^{-1}(w)\}$$

for all words $w_2 \in \Sigma_2^*$ and for all languages $L \subseteq \Sigma_2^*$. ■

Example 2.5 Consider the alphabet $\Sigma = \{a, b\}$. The size of Σ is $|\Sigma| = 2$. The second power Σ^2 and the iteration Σ^* is given by

$$\Sigma^2 = \{aa, ab, ba, bb\} \qquad \text{and} \qquad \Sigma^* = \{\varepsilon, a, b, aa, ab, ba, bb, \ldots\}.$$

The string $w = abbab$ is a word over Σ with length $|w| = 5$. The number of occurrences of the symbol a in w is $\#_a w = 2$, and of the symbol b in w is $\#_b w = 3$. The subwords ab and abb are prefixes of w and the subwords ab and bab are suffixes of w. bb is a subword of w, but neither a prefix nor a suffix. The set

$$L_1 = \Sigma^5 = \{w \in \Sigma^* \mid |w| = 5\}$$

is a finite and ε-free language over Σ because $|L_1| = |\Sigma|^5 = 2^5$ and $\varepsilon \notin L_1$ since $|\varepsilon| = 0 \neq 5$. The set

$$L_2 = \{ab\}^* = \{(ab)^i \mid i \in \mathbb{N}_0\}$$

is a language over Σ, too, but neither finite nor ε-free. Since at least $w = abbab \notin L_2$, $L_2 \subsetneqq \Sigma^*$ holds.

The mapping $h : \Sigma \to \Sigma'^*$ with $\Sigma' = \{c, d\}$ and $h(a) = c$, $h(b) = cd$ defines an ε-free homomorphism $h : \Sigma^* \to \Sigma'^*$. For example

$$h(abbab) = c\,(cd)^2\,c\,cd.$$

The mapping $\sigma : \Sigma \to \wp(\Sigma'^*)$ with $\sigma(a) = L_1 = \{a, b\}^5$ and $\sigma(b) = L_2 = \{ab\}^*$ defines a substitution $\sigma : \Sigma^* \to \wp(\Sigma'^*)$ which neither is finite nor ε-free because L_2 neither is finite nor ε-free. For example

$$\sigma(abbab) = L_1(L_2)^2 L_1 L_2.$$

Applying h and σ to the languages L_1 and L_2 above one obtains

$$\begin{aligned} h(L_1) &= \{h(a), h(b)\}^5 = \{c, cd\}^5, & h(L_2) &= \{h(a)\,h(b)\}^* = \{c\,cd\}^*, \\ \sigma(L_1) &= (\sigma(a) \cup \sigma(b))^5 = (L_1 \cup L_2)^5, & \sigma(L_2) &= (\sigma(a)\sigma(b))^* = (L_1 L_2)^*. \end{aligned}$$

Consider another ε-free homomorphism $g : \Sigma'^* \to \Sigma^*$ which is defined by $g(a) = c$ and $g(b) = cc$. Then the inverse homomorphism g^{-1} delivers for example

$$\begin{aligned} g^{-1}(cc) &= \{aa, b\}, \\ g^{-1}(cd) &= \emptyset, \\ g^{-1}(\varepsilon) &= \{\varepsilon\}. \ \blacksquare \end{aligned}$$

The main objects of study in formal language theory are finite specifications of infinite languages. All the specifications considered in this work are based upon so-called *rewriting systems*.

Definition 2.6 A *rewriting system* is a pair $G = (\Sigma, P)$ where Σ is an alphabet and $P \subseteq \Sigma^* \times \Sigma^*$ is a finite set of so-called *rewriting rules* or *productions*. A production $(v, w) \in P$ is also written as $v \to w$. The binary *yield relation* $\Longrightarrow_G$ on the set Σ^* is defined as follows:

$$\alpha \Longrightarrow_G \beta$$

holds if and only if there exist words $x_1, x_2, v, w \in \Sigma^*$ such that

$$\alpha = x_1 v x_2, \qquad \beta = x_1 w x_2, \qquad \text{and} \qquad v \to w$$

is a production in P. The relation $\Longrightarrow_G$ simply is written as $\Longrightarrow$ if it is unambiguous which rewriting system is used. The *transitive closure* or the *reflexive transitive closure* of $\Longrightarrow$ is denoted by $\Longrightarrow^+$ or $\Longrightarrow^*$, respectively. ■

A special kind of rewriting system is a *phrase structure grammar* or, simply, *grammar*.

Definition 2.7 A *grammar* is defined as an ordered quadruplet $G = (\Sigma, P, S, \Delta)$ where (Σ, P) is a rewriting system, $\Delta \subsetneqq \Sigma$ is the alphabet of *terminals*, $\Sigma \setminus \Delta$ is the alphabet of *non-terminals*, $S \in \Sigma \setminus \Delta$ is the *initial* non-terminal symbol, and each production $v \to w$ in P contains at least one non-terminal symbol on its left-hand side v. Let $\Longrightarrow_G$ be the yield relation of the underlying rewriting system (Σ, P). Then the *language* $L(G)$ *generated by* G is defined as

$$L(G) = E(G) \cap \Delta^* \qquad \text{where} \qquad E(G) = \{w \mid S \Longrightarrow_G^* w\}. \blacksquare$$

In other words, $L(G)$ denotes the set of all *terminal words* $w \in \Delta^*$ which can be derived from S according to the rewriting rules of G. The grammars are typed according to the shapes of their rewriting rules.

Definition 2.8 A grammar $G = (\Sigma, P, S, \Delta)$ is called to be of *type* i, for $i = 0, 1, 2, 3$, if P satisfies restriction (i), as given below:

(0) No restrictions.

(1) Each production in P is of the form

$$w_1 A w_2 \to w_1 w w_2 \qquad \text{where } w_1, w_2 \in \Sigma^*,\ A \in \Sigma \setminus \Delta,\ w \in \Sigma^+$$

with the exception of the production $S \to \varepsilon$ which is allowed only if S does not occur on the right-hand side of any production in P.

(2) Each production in P is of the form

$$A \to w \qquad \text{where } A \in \Sigma \setminus \Delta,\ w \in \Sigma^*.$$

(3) Each production in P is of one of the two forms

$$A \to Bw \text{ or } A \to w \qquad \text{where } A, B \in \Sigma \setminus \Delta,\ w \in \Delta^*.$$

A language L is called to be of *type* i, for $i = 0, 1, 2, 3$, if L is generated by a grammar of type i. The set or *family of all languages of type* i, for $i = 0, 1, 2, 3$, is denoted by $\mathcal{L}_i$. ■

Thus, every language generated by a grammar is of type 0.

Languages of type 0 are also called to be *recursively enumerable* which means that for each language L of type 0 there always exists an algorithm which enumerates exactly all words of L, allowing for repetitions (see [29] for the outline of a proof). If there exists an algorithm which can decide whether or not an arbitrary word belongs to the language L, then L is called to be *recursive*. The family of all recursively enumerable languages is termed $\mathcal{L}(\text{re})$ and the family of all recursive languages is termed $\mathcal{L}(\text{rec})$.

Grammars of type 1, 2, or 3 as well as their generated languages are also called *context-sensitive*, *context-free*, or *regular*, respectively. Therefore, the family $\mathcal{L}_i$, for $i = 1, 2, 3$, also is denoted by $\mathcal{L}(\text{cs})$, $\mathcal{L}(\text{cf})$, and $\mathcal{L}(\text{reg})$, respectively. Furthermore, the family of all finite languages is termed $\mathcal{L}(\text{fin})$.

These families form a strictly increasing hierarchy, the so-called *extended Chomsky Hierarchy* (see [41]) which is of high importance to formal language theory:

$$\mathcal{L}(\text{fin}) \subsetneqq \mathcal{L}_3 = \mathcal{L}(\text{reg}) \subsetneqq \mathcal{L}_2 = \mathcal{L}(\text{cf}) \subsetneqq \mathcal{L}_1 = \mathcal{L}(\text{cs}) \subsetneqq \mathcal{L}(\text{rec}) \subsetneqq \mathcal{L}_0 = \mathcal{L}(\text{re}) \qquad (2.1)$$

Two specifications of a language are called to be *equivalent* if they specify the same language. Consider e.g. the strict inclusion $\mathcal{L}_2 \subsetneqq \mathcal{L}_1$ of (2.1) which is logically equivalent to the conjunction of the inclusion $\mathcal{L}_2 \subseteq \mathcal{L}_1$ and the non-inclusion $\mathcal{L}_1 \not\subseteq \mathcal{L}_2$.

On the one hand, the inclusion $\mathcal{L}_2 \subseteq \mathcal{L}_1$ means that for each grammar G_2 of type 2 there always exists an equivalent grammar G_1 of type 1, i.e. with $L(G_2) = L(G_1)$. On the other hand, the non-inclusion $\mathcal{L}_1 \not\subseteq \mathcal{L}_2$ means that there exists a grammar G_1' of type 1 which is not equivalent to any of the grammars of type 2.

Example 2.9 The grammar $G = (\Sigma, P, S, \Delta)$ with $\Delta = \{a, b\}$, $\Sigma = \Delta \cup \{S\}$, and $P = \{S \to Sab,\ S \to \varepsilon\}$ generates the language $L(G) = \{ab\}^*$. Grammar G is of type 3 or regular and thus, by definition, it also is of type 2 and type 0. Therefore, also $L(G)$ at least is of the types 0, 2, and 3. Grammar G is not context-sensitive or of type 1 since, on the one hand, the initial symbol S is on the right-hand side of the production $S \to Sab$, but on the other hand, $S \to \varepsilon$ is a production of P, too.

However, since $L(G)$ is of type 2 the Chomsky Hierarchy implies that $L(G)$ also is of type 1 and thus, there also exists a grammar G' generating $L(G)$. For example, $G' = (\Sigma', P', S, \Delta)$ with $\Sigma' = \Sigma \cup \{T\}$ and $P' = \{S \to \varepsilon,\ S \to T,\ T \to Tab,\ T \to ab\}$ is a context-sensitive grammar which even is context-free with $L(G') = L(G) = \{ab\}^*$. Thus, $L(G)$ is of all the four types. ■

In formal language theory it is of further interest which kinds of operations, applied to formal languages of a certain type, result in a language of the same type. In general, a family $\mathcal{F}$ of formal languages is called to be *closed with respect to operation* ρ if ρ is an *operation on* $\mathcal{F}$, i.e. the result of ρ applied to elements of $\mathcal{F}$ itself is an element of $\mathcal{F}$.

Definition 2.10 A family $\mathcal{F}$ of formal languages is termed *abstract family of languages (AFL)* if $\mathcal{F}$ contains a non-empty language and is closed with respect to each of the following five types of operations:

(a) union,
i.e. $L \cup L' \in \mathcal{F}$ for all $L, L' \in \mathcal{F}$,

(b) ε-free iteration,
i.e. $L^+ \in \mathcal{F}$ for all $L \in \mathcal{F}$,

(c) intersection with regular languages,
i.e. $L \cap R \in \mathcal{F}$ for all $L \in \mathcal{F}$ and $R \in \mathcal{L}(\text{reg})$,

(d) ε-free homomorphism,
i.e. $h(L) \in \mathcal{F}$ for all $L \in \mathcal{F}$ and all ε-free homomorphisms $h : \Sigma^* \to \Sigma'^*$ with $L \subseteq \Sigma^*$,

(e) inverse homomorphism,
i.e. $h^{-1}(L) \in \mathcal{F}$ for all $L \in \mathcal{F}$ and all arbitrary homomorphisms $h : \Sigma'^* \to \Sigma^*$ with $L \subseteq \Sigma^*$.

$\mathcal{F}$ is called *full AFL* if $\mathcal{F}$ is an AFL which is additionally closed with respect to

(d') arbitrary homomorphism,
i.e. $h(L) \in \mathcal{F}$ for all $L \in \mathcal{F}$ and all arbitrary homomorphisms $h : \Sigma^* \to \Sigma'^*$ with $L \subseteq \Sigma^*$. ■

Example 2.11 Each of the families $\mathcal{L}_i$, $i = 0, 1, 2, 3$, is an AFL. The families $\mathcal{L}_0$, $\mathcal{L}_2$, and $\mathcal{L}_3$ are full AFLs; $\mathcal{L}_1$ is not a full AFL (see [29], Theorem IV.1.5). ■

Lemma 2.12 Every AFL $\mathcal{F}$ is infinite over an infinite alphabet Σ, i.e. $|\mathcal{F}| = \infty$ and $|\Sigma| = \infty$ for every alphabet Σ with $\Sigma^* \supseteq \bigcup_{L \in \mathcal{F}} L$.

Proof: Let $\mathcal{F}$ be an AFL. Then Definition 2.10 implies the existence of a non-empty language $L_0 \in \mathcal{F}$, a word $w \in L_0$, and an alphabet Σ_0 with $L \subseteq \Sigma_0^*$.

Consider the homomorphisms $h_i : \Sigma_i^* \to \Sigma_0^*$ defined by $h_i(i) = w$ where $\Sigma_i = \{i\}$ for all $i \in \mathbb{N}$. Then on the one hand, all the languages $L_i = h_i^{-1}(L_0)$ are non-empty because of $\Sigma_i \subseteq L_i$ and on the other hand, they are pairwise disjoint because of $L_i \subseteq \Sigma_i^*$. Together, it follows that they are unequal by pairs. Since $\mathcal{F}$ is closed with respect to inverse homomorphism (see Definition 2.10(e)), $L_i \in \mathcal{F}$ holds for all $i \in \mathbb{N}$ and thus, $|\mathcal{F}| \geq |\mathbb{N}| = \infty$.

Let Σ be an alphabet with $\Sigma^* \supseteq \bigcup_{L \in \mathcal{F}} L$. Then the above definitions and arguments imply that $\Sigma^* \supseteq \bigcup_{i \in \mathbb{N}} L_i$ and thus, $\Sigma \supseteq \bigcup_{i \in \mathbb{N}} \Sigma_i = \mathbb{N}$. Therefore, $|\Sigma| \geq |\mathbb{N}| = \infty$ holds. ■

Definition 2.13 A family $\mathcal{F}$ of formal languages is termed *anti-AFL* if $\mathcal{F}$ is not closed with respect to any of the five types of operations (a)-(e) of Definition 2.10. ■

Note that in some literature an anti-AFL is defined as additionally not to be closed with respect to concatenation (see [26], e.g.). This definition is stronger than Definition 2.13. The motivation for the stronger definition might stem from the fact that every AFL is also closed with respect to concatenation (see [29], Theorem IV.1.1). An example for an anti-AFL is represented at the end of Section 2.2.

2.2 Lindenmayer Systems and Languages

The specification of languages by grammars as represented in the previous section bases on rewriting systems which sequentially rewrite one symbol after another. Stimulated by his knowledge regarding parallel development processes in nature, e.g. the cell growing of plants, the biologist and mathematician Aristide Lindenmayer invented the following modified rewriting systems in 1968 [16, 17] which base on homomorphisms and substitutions rewriting all occurrences of symbols in one step.

Definition 2.14 An ordered quadruplet $G = (\Sigma, H, \omega, \Delta)$ is called *ET0L system* if Σ is an alphabet, $\Delta \subseteq \Sigma$ (called *terminal alphabet*), $\omega \in \Sigma^*$ (called *initial word* or *axiom*), and H is a finite and non-empty set of finite and non-empty substitutions $h : \Sigma^* \to \wp(\Sigma^*)$ (called *tables*). A pair (a, w) is called a *production of table* h, also written as $a \xrightarrow{h} w$, if $a \in \Sigma$ and $w \in h(a)$. If $\Delta = \Sigma$, then Δ can be omitted and $G = (\Sigma, H, \omega)$ is called a *(non-extended) T0L* system. If $H = \{h\}$, then the singleton H can be replaced by the table h and $G = (\Sigma, h, \omega, \Delta)$ or $G = (\Sigma, h, \omega)$ is called *E0L* system or *0L* system, respectively. If each table $h \in H$ always maps to a singleton set, then each substitution h can be written as homomorphism on Σ^* and G is called *deterministic* or *(E)D(T)0L* system. If each table $h \in H$ is ε-free, then G is called *propagating* or *(E)P(D)(T)0L* system. ■

Note that the integer 0 in the above definition stands for the number of symbols or cells in the neighborhood of a cell a which influence the behavior of growing of cell a. Therefore, integer 0 means that all productions are context-free. By contrast, Lindenmayer systems with context-sensitive parallel rewriting are called *IL systems* where the letter *I* stands for the *interaction* between the cells of growing filamentous organisms (see [26] for an overview). A weaker form of interaction basing on 0L systems was introduced by k-limited 0L systems which are described in the next section.

Definition 2.15 The following abbreviations for special sets of type designators of 0L systems and languages are introduced:

(a) $\mathrm{TYPE}_{0\mathrm{L}} = \{\varepsilon, \mathrm{P}, \mathrm{D}, \mathrm{PD}\}$
representing all 4 possible types of 0L systems and languages.

(b) $\mathrm{TYPE}_{\mathrm{T0L}} = \mathrm{TYPE}_{0\mathrm{L}} \cdot \{\mathrm{T}, \varepsilon\}$
representing all 8 possible types of T0L systems and languages.

(c) $\mathrm{TYPE}_{\mathrm{E0L}} = \{\mathrm{E}, \varepsilon\} \cdot \mathrm{TYPE}_{\mathrm{0L}}$
representing all 8 possible types of E0L systems and languages.

(d) $\mathrm{TYPE}_{\mathrm{ET0L}} = \{\mathrm{E}, \varepsilon\} \cdot \mathrm{TYPE}_{\mathrm{T0L}}$
representing all 16 possible types of ET0L systems and languages. ■

Definition 2.16 Given an ET0L system $G = (\Sigma, H, \omega, \Delta)$, the binary *yield relation* $\Longrightarrow_G$ on the set Σ^* is defined as follows: $v \Longrightarrow_G w$ holds if there exists a table $h \in H$ such that $w \in h(v)$. Then this relation also is written as $v \overset{h}{\Longrightarrow}_G w$ or, if it is unambiguous, simply $v \Longrightarrow w$. The transitive or reflexive transitive closure of $\Longrightarrow_G$ is denoted by $\Longrightarrow_G^+$ or $\Longrightarrow_G^*$, respectively. Then the *language* $L(G)$ *generated by* G is defined as

$$L(G) = E(G) \cap \Delta^* \qquad \text{where} \qquad E(G) = \{w \mid \omega \Longrightarrow_G^* w\}. \blacksquare$$

Definition 2.17 Let $\tau \in \mathrm{TYPE}_{\mathrm{ET0L}}$. A language generated by a τ0L system is called a *τ0L language*. The family of all τ0L languages is denoted by $\mathcal{L}(\tau\mathrm{0L})$. ■

Example 2.18 (a) The PD0L system $G = (\{a\}, h, a)$ with $h(a) = a^2$ generates the language $L_1 = \{a^{2^n} \mid n \in \mathbb{N}_0\}$. Thus, $L_1 \in \mathcal{L}(\mathrm{PD0L})$.

(b) The language $L_2 = \{xwxwxw \mid w \in \{a, b\}^*\}$ is generated by the EPDT0L system $G = (\Sigma, H, \omega, \Delta)$ with $\Delta = \{x, a, b\}$, $\Sigma = \Delta \cup \{X, A, B\}$, $\omega = XXX$, and $H = \{h_1, h_2\}$, where

$$\begin{aligned} h_1(X) &= A, \quad h_1(A) = Xa, \quad h_1(B) = Xb, \\ h_2(X) &= B, \quad h_2(A) = x, \qquad h_2(B) = x, \end{aligned}$$

and $h_i(c) = c$ for $i = 1, 2$ and $c \in \Delta$. Thus, $L_2 \in \mathcal{L}(\mathrm{EPDT0L})$. ■

The definition of the families of Lindenmayer languages implies several trivial inclusions such as

$$\mathcal{L}(\mathrm{0L}) \subseteq \left\{ \begin{array}{c} \mathcal{L}(\mathrm{T0L}) \\ \mathcal{L}(\mathrm{E0L}) \end{array} \right\} \subseteq \mathcal{L}(\mathrm{ET0L}) \tag{2.2}$$

and

$$\mathcal{L}(\mathrm{PD0L}) \subseteq \left\{ \begin{array}{c} \mathcal{L}(\mathrm{P0L}) \\ \mathcal{L}(\mathrm{D0L}) \end{array} \right\} \subseteq \mathcal{L}(\mathrm{0L}) \tag{2.3}$$

where (2.2) analogously holds for the corresponding families of P0L, D0L, and PD0L languages, and (2.3) for the corresponding families of E0L, T0L, and ET0L languages. Altogether, these inclusion relations between the 16 types of families of

ET0L languages form a 4-dimensional directed cube-graph (*hyper cube*) as represented in Figure 2.1 where $A \to B$ stands for $A \subseteq B$.

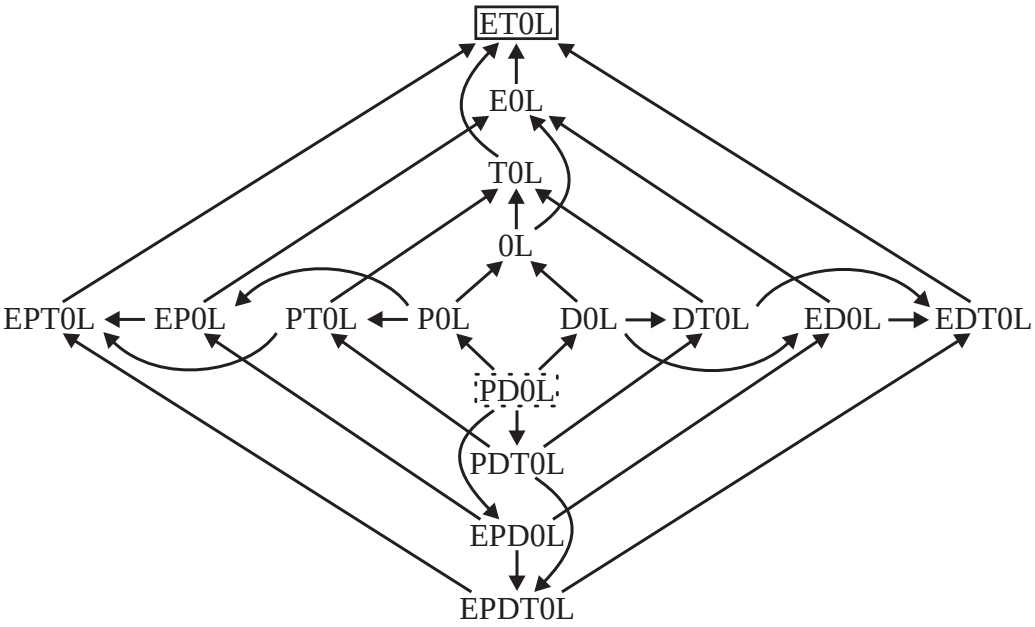

Figure 2.1. The *type hyper cube* of the families of ET0L languages ($A \to B$ stands for $A \subseteq B$).

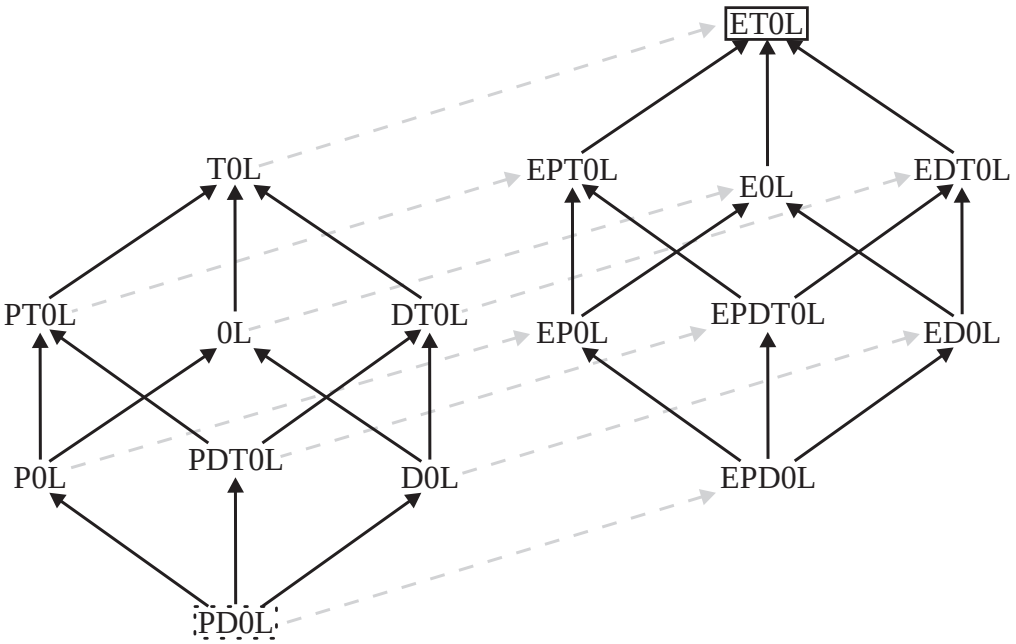

Figure 2.2. Two 3-dimensional cube-graphs included in the hyper cube of Figure 2.1 representing extended and non-extended language families.

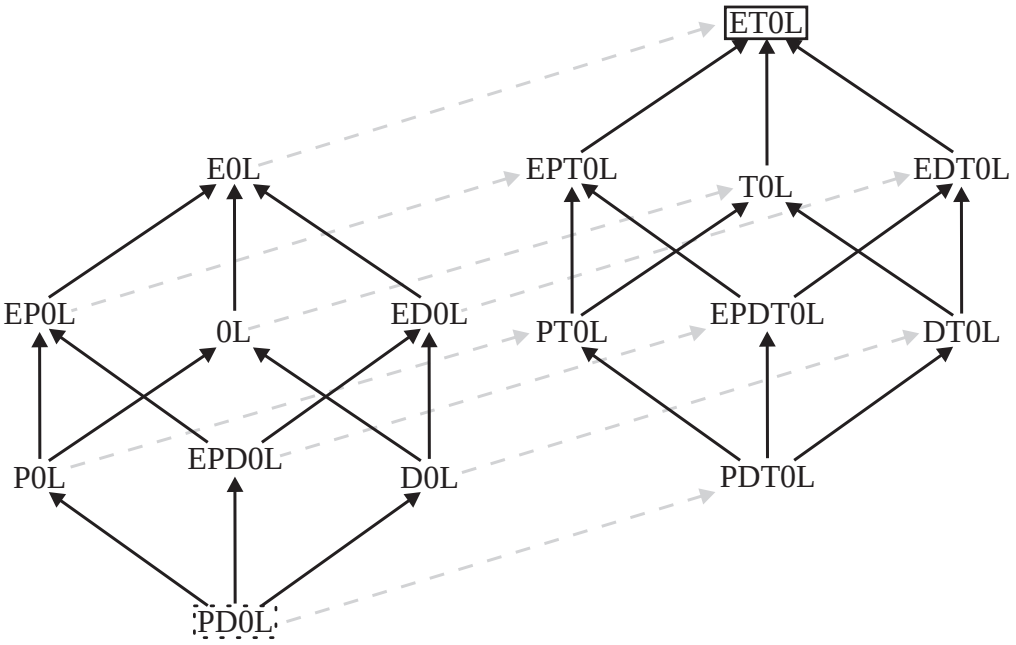

Figure 2.3. Two 3-dimensional cube-graphs included in the hyper cube of Figure 2.1 representing language families with and without tables.

The hyper cube-graph can be represented as two connected 3-dimensional cube-graphs in various manner as depicted for example in Figure 2.2 and Figure 2.3. The innermost cube-graph of Figure 2.1 represents inclusion relations between the eight types of families of non-extended T0L languages and corresponds with the left cube-graph of Figure 2.2. The outermost cube-graph of Figure 2.1 concerns the remaining eight types of families of ET0L languages and corresponds with the right cube-graph of Figure 2.2.

The arrows shown in Figure 2.1 (respectively Figure 2.2) are of two different types. The first type, in Figure 2.1 leading from the center of the cube to the outside (respectively the eight vertical and the eight dashed arrows of Figure 2.2), represents inclusions of type (2.2). The second type, in Figure 2.1 surrounding the center of the cube (respectively the 16 diagonal continuous arrows of Figure 2.2), represents inclusions of type (2.3). Thus, the smallest family $\mathcal{L}$(PD0L) which is included in each of the 16 families is placed on the bottom of the innermost ring (see the dashed box in Figure 2.1 or Figure 2.2, respectively) and the largest family $\mathcal{L}$(ET0L) which includes each of the 16 families is placed on top of the outermost ring (see the box on top of Figure 2.1 or Figure 2.2, respectively).

These facts are noted in the following Definition 2.19 and Lemma 2.20.

Definition 2.19 The following abbreviations for special sets of pairs of type designators of 0L systems and languages are introduced:

(a) $\mathrm{CUBE}_{0\mathrm{L}} = \{(\tau,\tau) \mid \tau \in \mathrm{TYPE}_{0\mathrm{L}}\} \cup \{(\mathrm{PD},\mathrm{P}),(\mathrm{PD},\mathrm{D}),(\mathrm{PD},\varepsilon),(\mathrm{P},\varepsilon),(\mathrm{D},\varepsilon)\}$
representing all the 9 trivial inclusions between two (not necessary different) families of 0L languages according to Figure 2.1.

(b) $\mathrm{CUBE}_{\mathrm{T0L}} = \mathrm{CUBE}_{0\mathrm{L}} \cup \{(\tau_1,\tau_2\mathrm{T}),(\tau_1\mathrm{T},\tau_2\mathrm{T}) \mid (\tau_1,\tau_2) \in \mathrm{CUBE}_{0\mathrm{L}}\}$
representing all the $27 = 3 \cdot |\mathrm{CUBE}_{0\mathrm{L}}|$ trivial inclusions between two (not necessary different) families of T0L languages according to Figure 2.1.

(c) $\mathrm{CUBE}_{\mathrm{E0L}} = \mathrm{CUBE}_{0\mathrm{L}} \cup \{(\tau_1,\mathrm{E}\tau_2),(\mathrm{E}\tau_1,\mathrm{E}\tau_2) \mid (\tau_1,\tau_2) \in \mathrm{CUBE}_{0\mathrm{L}}\}$
representing all the $27 = 3 \cdot |\mathrm{CUBE}_{0\mathrm{L}}|$ trivial inclusions between two (not necessary different) families of E0L languages according to Figure 2.1.

(d) $\mathrm{CUBE}_{\mathrm{ET0L}} = \mathrm{CUBE}_{\mathrm{T0L}} \cup \{(\tau_1,\mathrm{E}\tau_2),(\mathrm{E}\tau_1,\mathrm{E}\tau_2) \mid (\tau_1,\tau_2) \in \mathrm{CUBE}_{\mathrm{T0L}}\}$
representing all the $81 = 3 \cdot |\mathrm{CUBE}_{\mathrm{T0L}}|$ trivial inclusions between two (not necessary different) families of ET0L languages according to Figure 2.1. ■

Lemma 2.20 $\mathcal{L}(\tau_1 0\mathrm{L}) \subseteq \mathcal{L}(\tau_2 0\mathrm{L})$ for all $(\tau_1,\tau_2) \in \mathrm{CUBE}_{\mathrm{ET0L}}$. ■

Beside these trivial inclusions the following results concerning families of ET0L languages are known:

$$\mathcal{L}(\mathrm{cf}) \subsetneqq \mathcal{L}(\mathrm{E0L}) = \mathcal{L}(\mathrm{EP0L}) \subsetneqq \mathcal{L}(\mathrm{ET0L}) \subsetneqq \mathcal{L}(\mathrm{cs}),$$

$$\mathcal{L}(0\mathrm{L}) \subsetneqq \mathcal{L}(\mathrm{E0L}), \quad \text{and} \quad \mathcal{L}(\mathrm{EDT0L}) \subsetneqq \mathcal{L}(\mathrm{ET0L}).$$

The family $\mathcal{L}(0\mathrm{L})$ is incomparable to each of the families $\mathcal{L}(\mathrm{cf})$, $\mathcal{L}(\mathrm{reg})$, and $\mathcal{L}(\mathrm{fin})$, but it is not disjoint to them. Moreover, $\mathcal{L}(0\mathrm{L})$ is an anti-AFL and additionally not closed with respect to concatenation. But it is closed with respect to iteration as well as to the operation called *mirror image* which reverses the order of letters of a given word. The family $\mathcal{L}(\mathrm{E0L})$ is closed with respect to union, concatenation, and ε-free iteration. The family $\mathcal{L}(\mathrm{ET0L})$ is even a full AFL. For further reading about ET0L systems and languages see [26, 29].

2.3 k-limited Lindenmayer Systems and Languages

The original Lindenmayer systems describe unlimited parallel cell growing without any interactions between the cells. Thus, they allow permanent exponential cell

growing as visualized in Figure 1.2 on page 4 and as demonstrated by the PD0L language $L_1 = \{a^{2^n} \mid n \in \mathbb{N}_0\}$ of example 2.18. At least partly, this does not correspond to the biological reality. A more realistic approach is represented by k-limited Lindenmayer systems which were introduced by Wätjen in 1988 [36]. The rewriting of these systems neither is fully parallel like ET0L systems nor sequential like grammars. They describe a kind of gradual cell growing where in every step the growing of each cell-type is restricted by a reservoir of food of a fixed common size k. Via this limitation of the growing for each cell-type, there is a weak form of interaction between the cells of the same type regardless of the context-free rewriting of the underlying 0L system.

Definition 2.21 Given an integer $k \in \mathbb{N}$ and a finite and non-empty substitution h on an alphabet Σ, the mapping $h_k : \Sigma^* \to \wp(\Sigma^*)$ is called the *k-limitation of h*, where $h_k(w)$ is defined as the set of all words that arise from the word w by simultaneously rewriting all, but at most k occurrences of each symbol $a \in \Sigma$ by an arbitrary word $v \in h(a)$.

For $w = a_1 \cdots a_n$ with $a_i \in \Sigma$ for $i = 1, \ldots, n$ and $n \geq 0$, and $I_a(w) = \{i \mid a_i = a\}$ for all $a \in \Sigma$, this definition also is given by the formula

$$h_k(w) = \{v_1 \cdots v_n \mid \forall_{a\in\Sigma} \quad \exists_{R_a \subseteq I_a(w)} \\ (|R_a| = \min\{\#_a w,\, k\} \wedge \forall_{i\in R_a} v_i \in h(a) \wedge \forall_{i \in I_a(w)\setminus R_a} v_i = a)\}. \blacksquare$$

Definition 2.22 An ordered quintuplet $G = (\Sigma, H, \omega, \Delta, k)$ is called *k-limited ET0L system* or, briefly, *klET0L system* if $G' = (\Sigma, H, \omega, \Delta)$ is an ET0L system and k is a positive integer. System G is called *$k l\tau$0L* system if G' is a τ0L system, respectively for $\tau \in \mathrm{TYPE}_{\mathrm{ET0L}}$ (see Definition 2.14). $\blacksquare$

Definition 2.23 Given a klET0L system $G = (\Sigma, H, \omega, \Delta, k)$, the binary *yield relation* $\Longrightarrow_G$ on the set Σ^* is defined as follows: $v \Longrightarrow_G w$ holds if there exists a table $h \in H$ such that $w \in h_k(v)$. Then this statement is also written as $v \overset{h}{\Longrightarrow}_G w$ or, if it is unambiguous, just as $v \Longrightarrow w$. The transitive or reflexive transitive closure of $\Longrightarrow_G$ is denoted by $\Longrightarrow_G^+$ or $\Longrightarrow_G^*$, respectively. Then the *language $L(G)$ generated by G* is defined as

$$L(G) = E(G) \cap \Delta^* \qquad \text{where} \qquad E(G) = \{w \mid \omega \Longrightarrow_G^* w\}. \blacksquare$$

Definition 2.24 Let $\tau \in \mathrm{TYPE}_{\mathrm{ET0L}}$. A language generated by a $k\mathrm{l}\tau$0L system is called *$k l\tau$0L language*. For fixed $k \in \mathbb{N}$, the family of all $k\mathrm{l}\tau$0L languages is denoted

by $\mathcal{L}(k\mathrm{l}\tau\mathrm{0L})$, and

$$\mathcal{L}(\mathrm{l}\tau\mathrm{0L}) = \bigcup_{k\in\mathbb{N}} \mathcal{L}(k\mathrm{l}\tau\mathrm{0L})$$

denotes the family of all such limited languages. ■

Example 2.25 (a) For every $k \in \mathbb{N}$, the klPD0L system $G = (\{a\}, h, a, k)$ with $h(a) = a^2$ generates the language

$$L_k = \{a^{2^n} \mid n = 0, \ldots, N\} \cup \{a^{2^N+kn} \mid n \in \mathbb{N}\},$$

where $N = \lfloor \log_2 k \rfloor + 1$. More generally, for $h(a) = a^i$ with $i \geq 2$, system G generates the language

$$L_{k,i} = \{a^{i^n} \mid n = 0, \ldots, N\} \cup \{a^{i^N+(i-1)kn} \mid n \in \mathbb{N}\},$$

where $N = \lfloor \log_i k \rfloor + 1$. Thus, $L_{k,2} = L_k$ and $L_{k,i} \in \mathcal{L}(k\mathrm{lPD0L})$, for all $k, i \in \mathbb{N}$ with $i \geq 2$.

(b) For every $k \in \mathbb{N}$, the language $L_0 = \{a^{2^n} \mid n \in \mathbb{N}_0\}$ is generated by each of the klEPDT0L systems $G_k = (\Sigma, H, a, \{a\}, k)$ with $\Sigma = \{a, X, Y, F\}$ and $H = \{h_1, h_2, h_3\}$, where

$$\begin{array}{llll} h_1(a) = X, & h_1(X) = X, & h_1(Y) = F, & h_1(F) = F, \\ h_2(a) = F, & h_2(X) = Y, & h_2(Y) = Y, & h_2(F) = F, \\ h_3(a) = a, & h_3(X) = F, & h_3(Y) = a^2, & h_3(F) = F. \end{array}$$

Each of the systems G_k works as follows: The non-terminal symbol F (*failure* symbol) never can be removed again once it has been introduced. Its occurrence thus indicates that the generation of a terminal word has failed. Therefore, in order to avoid the failure symbol F, the tables h_1, h_2, h_3 have to be applied to the axiom $a = a^{2^0}$ or another generated terminal word a^{2^n}, for $n \in \mathbb{N}_0$, in the following order:

$$a^{2^n} \overset{h_1}{\Longrightarrow}{}^*_G X^{2^n} \overset{h_2}{\Longrightarrow}{}^*_G Y^{2^n} \overset{h_3}{\Longrightarrow}{}^*_G a^{2^{n+1}}.$$

Thus, $L_0 \in \mathcal{L}(k\mathrm{lEPDT0L})$ for every $k \in \mathbb{N}$. ■

The definition of the families $\mathcal{L}((k)\mathrm{l}\tau\mathrm{0L})$ of k-limited Lindenmayer languages for $\tau \in \mathrm{TYPE}_{\mathrm{ET0L}}$, directly implies trivial inclusions which are analogous to inclusions (2.2) and (2.3) on page 16 of the non-limited case. More precisely, one obtains a *type* as shown in Figure 2.1 on page 17 for the 16 types of families of k-limited languages for every fixed limit $k \in \mathbb{N}$ as well as for the case that k is omitted everywhere. These facts are summarized in the following Lemma 2.26.

Lemma 2.26 $\mathcal{L}(\mathrm{l}\tau_1\mathrm{0L}) \subseteq \mathcal{L}(\mathrm{l}\tau_2\mathrm{0L})$ as well as $\mathcal{L}(k\mathrm{l}\tau_1\mathrm{0L}) \subseteq \mathcal{L}(k\mathrm{l}\tau_2\mathrm{0L})$ for all $k \in \mathbb{N}$ and $(\tau_1, \tau_2) \in \mathrm{CUBE}_{\mathrm{ET0L}}$. ■

In [36] Wätjen strengthened these relations by proving several strict inclusions such as

$$\mathcal{L}(k\mathrm{lPD0L}) \subsetneqq \left\{ \begin{array}{c} \mathcal{L}(k\mathrm{lP0L}) \\ \mathcal{L}(k\mathrm{lD0L}) \end{array} \right\} \subsetneqq \mathcal{L}(k\mathrm{l0L})$$

and, moreover, that the families $\mathcal{L}(k\mathrm{lP0L})$ and $\mathcal{L}(k\mathrm{lD0L})$ are incomparable to each other, but not disjoint, for every $k \in \mathbb{N}$. Analogous results hold for the corresponding families of l0L, lT0L, and of klT0L languages, for all $k \geq 2$, as well as for the corresponding families of lE0L and klE0L languages, for all $k \in \mathbb{N}$. Further results proved by Wätjen in [36] and [42] are

$$\mathcal{L}(1\mathrm{lET0L}) = \mathcal{L}(\mathrm{lET0L}) \quad \text{and} \quad \mathcal{L}(\mathrm{ET0L}) \subsetneqq \mathcal{L}(k\mathrm{lET0L})$$

as well as

$$\mathcal{L}(k\mathrm{lED0L}) \subsetneqq \mathcal{L}(\mathrm{cs}) \quad \text{and} \quad \mathcal{L}(k\mathrm{lEPT0L}) \subseteq \mathcal{L}(\mathrm{cs}),$$

for all $k \in \mathbb{N}$. The latter result was strengthened by Dassow in [4] for $k = 1$ to be a strict inclusion

$$\mathcal{L}(1\mathrm{lEPT0L}) \subsetneqq \mathcal{L}(\mathrm{cs}).$$

In contrast to the inclusion relations above, for all $\tau_1, \tau_2 \in \mathrm{TYPE}_{\mathrm{T0L}}$ each family $\mathcal{L}((k)\mathrm{l}\tau_1\mathrm{0L})$, with $k \in \mathbb{N}$, is incomparable to each family $\mathcal{L}((k')\mathrm{l}\tau_2\mathrm{0L})$, with $k' \neq k$, and to each family $\mathcal{L}(\tau_2\mathrm{0L})$ of non-limited Lindenmayer languages, as well as to $\mathcal{L}(\mathrm{cf})$, $\mathcal{L}(\mathrm{reg})$, and $\mathcal{L}(\mathrm{fin})$.

The closure properties of the families of k-limited Lindenmayer languages are as various as in the non-limited case. On the one hand, each family $\mathcal{L}((k)\mathrm{l}\tau\mathrm{0L})$, with $k \in \mathbb{N}$ and $\tau \in \mathrm{TYPE}_{\mathrm{T0L}}$, is an anti-AFL. On the other hand, for every $k \in \mathbb{N}$, the family $\mathcal{L}(k\mathrm{lET0L})$ almost is a full AFL since it is closed with respect to union, concatenation, homomorphism, ε-free iteration, and substitution. If it is also closed with respect to intersection with regular languages it is a full AFL. But this still is an open question.

It is also still unsolved whether or not the families $\mathcal{L}(k\mathrm{lET0L})$ are incomparable to the families $\mathcal{L}(k'\mathrm{lET0L})$, for $k' \neq k$ with $k, k' \neq 1$, or to the family $\mathcal{L}(\mathrm{cs})$. For the latter case, Fernau proved $\mathcal{L}(k\mathrm{lET0L}) \not\subseteq \mathcal{L}(\mathrm{rec})$ in [9], for all $k \in \mathbb{N}$, which implies

$$\mathcal{L}(k\mathrm{lET0L}) \not\subseteq \mathcal{L}(\mathrm{cs})$$

because of $\mathcal{L}(\mathrm{cs}) \subsetneqq \mathcal{L}(\mathrm{rec})$ according to the Chomsky Hierarchy (2.1).

2.4 Multi-limited Lindenmayer Systems and Languages

As mentioned in the previous section, k-limited Lindenmayer systems are modeling a kind of limited cell growing whereat the growing of all cells of the same type respectively is limited by a common reservoir of food of the constant size k. In order to model a kind of cell growing which is more flexible, multi-limited Lindenmayer systems were suggested by Gärtner in [12]. These systems generalize k-limited L systems by providing a limiting function $\kappa : \Sigma \to \mathbb{N}$, instead of the constant integer $k \in \mathbb{N}$, attaching to each cell-type $a \in \Sigma$ a reservoir of food of individual size $\kappa(a)$.

Multi-limited Lindenmayer systems are introduced in this section and investigated in the following chapters. For the visualization of a word generated by a multi-limited Lindenmayer system see Figure 1.3 on page 5.

Definition 2.27 Given a function $\kappa : \Sigma \to \mathbb{N}$ and a finite and non-empty substitution h on an alphabet Σ, the mapping $h_\kappa : \Sigma^* \to \wp(\Sigma^*)$ is called the *κ-limitation of h*, where $h_\kappa(w)$ is defined as the set of all words which arise from the word w by simultaneously rewriting all, but at most $\kappa(a)$ occurrences of each symbol $a \in \Sigma$ by an arbitrary word $v \in h(a)$.

For $w = a_1 \cdots a_n$ with $a_i \in \Sigma$ for $i = 1, \ldots, n$ and $n \geq 0$, and $I_a(w) = \{i \mid a_i = a\}$ for all $a \in \Sigma$, this definition also is given by the formula

$$h_\kappa(w) = \{v_1 \cdots v_n \mid \forall_{a\in\Sigma} \quad \exists_{R_a \subseteq I_a(w)} \\ (|R_a| = \min\{\#_a w,\, \kappa(a)\} \wedge \forall_{i\in R_a} v_i \in h(a) \wedge \forall_{i \in I_a(w)\setminus R_a} v_i = a)\}. \blacksquare$$

Definition 2.28 An ordered quintuplet $G = (\Sigma, H, \omega, \Delta, \kappa)$ is called a *κ-multi-limited ET0L system*, or briefly *κmlET0L system* or just *mlET0L system*, if $G' = (\Sigma, H, \omega, \Delta)$ is an ET0L system and $\kappa : \Sigma \to \mathbb{N}$ is a mapping, the so-called *limiting function*. For each type $\tau \in \mathrm{TYPE}_{\mathrm{ET0L}}$, the system G is called a *κmlτ0L* system if G' is a τ0L system. $\blacksquare$

Definition 2.29 Given a κmlET0L system $G = (\Sigma, H, \omega, \Delta, \kappa)$, the binary *yield relation* $\Longrightarrow_G$ on the set Σ^* is defined as follows: $v \Longrightarrow_G w$ holds if there exists a table $h \in H$ such that $w \in h_\kappa(v)$. This statement also is written as $v \overset{h}{\Longrightarrow}_G w$ or just as $v \Longrightarrow w$, if it is unambiguous. The transitive or reflexive transitive closure

of $\Longrightarrow_G$ is denoted by $\Longrightarrow_G^+$ or $\Longrightarrow_G^*$, respectively. The *language* $L(G)$ *generated by* G is defined as

$$L(G) = E(G) \cap \Delta^* \qquad \text{where} \qquad E(G) = \{w \mid \omega \Longrightarrow_G^* w\}. \blacksquare$$

Example 2.30 Definition 2.29 immediately implies that each klET0L system $G_k = (\Sigma, H, \omega, \Delta, k)$, $k \in \mathbb{N}$, can be written as the equivalent κmlET0L system $G_\kappa = (\Sigma, H, \omega, \Delta, \kappa)$, where $\kappa(x) = k$ for all $x \in \Sigma$. Thus, by Example 2.25(b), the language $L_0 = \{a^{2^n} \mid n \in \mathbb{N}_0\}$ also is generated by a multi-limited L system. $\blacksquare$

Definitions 2.27–2.29 describe a mechanism which generates words in a certain new way. The mechanism is coded by the five parameters Σ, H, ω, Δ, and κ. Since in contrast to k-limited L systems, the definition of the limiting function $\kappa : \Sigma \to \mathbb{N}$ depends on the alphabet Σ, the limiting function κ itself is not suitable to characterize the specific part of the mechanism of multi-limited L systems.

For example, since the language $L_0 = \{a^{2^n} \mid n \in \mathbb{N}_0\}$ of Example 2.30 is generated by a multi-limited L system G, the language $L_0' = \{b^{2^n} \mid n \in \mathbb{N}_0\}$ also is generated by a similar multi-limited L system G' which uses almost the same word generating mechanism as G does.

Therefore, instead of the limiting function κ itself, in the following categorization of multi-limited Lindenmayer languages the image set $\mathrm{Im}(\kappa)$ is used. The set $\mathrm{Im}(\kappa)$ consists of all limits which are assigned to a cell-type a, regardless of the concrete assignment $a \mapsto \kappa(a)$. Thus, the definition of $\mathrm{Im}(\kappa)$ does not depend on the alphabet Σ.

Definition 2.31 Let $\tau \in \mathrm{TYPE}_{\mathrm{ET0L}}$. The language generated by a κmlτ0L system is called a *$\kappa ml\tau 0L$ language*. For each non-empty set $K \subseteq \mathbb{N}$ with $\mathrm{Im}(\kappa) \subseteq K$, they are also called a *$K ml\tau 0L$* system respectively language. In this case, the set K is called the *set of permitted limits* of the system. For fixed $K \subseteq \mathbb{N}$, the family of all Kmlτ0L languages is denoted by $\mathcal{L}(K\mathrm{ml}\tau\mathrm{0L})$, and

$$\mathcal{L}(\mathrm{ml}\tau\mathrm{0L}) = \bigcup_{\emptyset \neq K \subseteq \mathbb{N}} \mathcal{L}(K\mathrm{ml}\tau\mathrm{0L})$$

denotes the family of all such multi-limited languages. $\blacksquare$

Example 2.32 (a) Every klET0L system $G = (\Sigma, H, \omega, \Delta, k)$ can be written as the equivalent KmlET0L system $G' = (\Sigma, H, \omega, \Delta, \kappa)$, and vice versa, if $K = \mathrm{Im}(\kappa) = \{k\}$.

(b) The κmlPD0L system $G = (\{a, b\}, h, ab, \kappa)$ with $h(a) = a^2$, $h(b) = b^2$, and $\kappa(a) = 3$, $\kappa(b) = 8$, generates the language

$$L = \{ab,\, a^2b^2,\, a^4b^4,\, a^7b^8\} \cup \{a^{7+3n}b^{8+8n} \mid n \in \mathbb{N}\}.$$

Thus, $L \in \mathcal{L}(\{3, 8\}\text{mlPD0L})$. ■

The language represented in Example 2.32 (b) consists of words which grow in each derivation step by at most $\kappa(a)+\kappa(b) = 11$ symbols. The underlying non-limited 0L system, in contrary, shows an exponential growing. For the general case of mlET0L systems the following Theorem of bounded growth obviously holds.

Theorem 2.33 (Bounded Growth Theorem) Let $G = (\Sigma, H, \omega, \Delta, \kappa)$ be a KmlET0L system with non-empty set $K \subseteq \mathbb{N}$, $s(a) = \max\{|w| \mid w \in h(a),\, h \in H\}$ for $a \in \Sigma$, $c_1(G) = \sum_{a\in\Sigma} \kappa(a)$, and $c_2(G) = \sum_{a\in\Sigma} \kappa(a) \cdot (s(a) - 1)$. Then for all $w_1, w_2 \in \Sigma^*$ with $w_1 \Longrightarrow w_2$ it holds that

$$|w_1| - c_1(G) \leq |w_2| \leq |w_1| + c_2(G). \blacksquare$$

An immediate consequence of Theorem 2.33 concerning mlT0L systems is the following Corollary 2.34. In order to formulate this corollary more generally, it also uses a small lemma, Lemma 2.35, which is proved subsequently.

Corollary 2.34 The context-sensitive language $\{a^{2^n} \mid n \in \mathbb{N}_0\}$ is not a member of $\mathcal{L}(K\text{ml}\tau\text{0L})$ for all non-empty sets $K \subseteq \mathbb{N}$ and $\tau \in \text{TYPE}_{\text{T0L}}$. ■

In the sequel, two immediate results are derived from Definition 2.31. The first, Lemma 2.35 and its corollaries, consider inclusion relations depending on the set K of permitted limits. The second, Theorem 2.38, uses these results for describing some type depending inclusion relations which are analogous to the inclusion relations of ET0L languages, i.e. inclusions (2.2) and (2.3) on page 16.

Lemma 2.35 For all non-empty sets $K_1, K_2 \subseteq \mathbb{N}$ and all $\tau \in \text{TYPE}_{\text{ET0L}}$, the following implication holds:

$$\text{if} \quad K_1 \subseteq K_2 \quad \text{then} \quad \mathcal{L}(K_1\text{ml}\tau\text{0L}) \subseteq \mathcal{L}(K_2\text{ml}\tau\text{0L}). \tag{2.4}$$

Proof: Let $\emptyset \neq K_1 \subseteq K_2 \subseteq \mathbb{N}$ and $\tau \in \text{TYPE}_{\text{ET0L}}$. Then by Definition 2.31, each K_1mlτ0L system G with limiting function κ also is a K_2mlτ0L system due to $\text{Im}(\kappa) \subseteq K_1 \subseteq K_2$. Thus, each K_1mlτ0L language also is a K_2mlτ0L language by Definition 2.31. ■

Corollary 2.36 For all non-empty sets $K_1, K_2 \subseteq \mathbb{N}$ and all $\tau \in \mathrm{TYPE}_{\mathrm{ET0L}}$ it holds that

$$\mathcal{L}(K_1\mathrm{ml}\tau\mathrm{0L}) \cup \mathcal{L}(K_2\mathrm{ml}\tau\mathrm{0L}) \subseteq \mathcal{L}((K_1 \cup K_2)\mathrm{ml}\tau\mathrm{0L})$$

as well as

$$\mathcal{L}(K_1\mathrm{ml}\tau\mathrm{0L}) \cap \mathcal{L}(K_2\mathrm{ml}\tau\mathrm{0L}) \supseteq \mathcal{L}((K_1 \cap K_2)\mathrm{ml}\tau\mathrm{0L}).$$

Proof: Consider arbitrary non-empty sets $K_1, K_2 \subseteq \mathbb{N}$ and $\tau \in \mathrm{TYPE}_{\mathrm{ET0L}}$. Since $K_1 \subseteq K_1 \cup K_2$ or $K_1 \supseteq K_1 \cap K_2$, respectively, Lemma 2.35 implies that

$$\mathcal{L}(K_1\mathrm{ml}\tau\mathrm{0L}) \subseteq \mathcal{L}((K_1 \cup K_2)\mathrm{ml}\tau\mathrm{0L}) \quad \text{and} \quad \mathcal{L}(K_1\mathrm{ml}\tau\mathrm{0L}) \supseteq \mathcal{L}((K_1 \cap K_2)\mathrm{ml}\tau\mathrm{0L}),$$

respectively. The exchanging of the sets K_1 and K_2 completes the proof. ■

Corollary 2.37 $\mathcal{L}(\mathbb{N}\mathrm{ml}\tau\mathrm{0L}) = \mathcal{L}(\mathrm{ml}\tau\mathrm{0L})$ for all $\tau \in \mathrm{TYPE}_{\mathrm{ET0L}}$.

Proof: Let $L \in \mathcal{L}(\mathrm{ml}\tau\mathrm{0L})$ for $\tau \in \mathrm{TYPE}_{\mathrm{ET0L}}$. Then by Definition 2.31 and Lemma 2.35, there exists a non-empty set $K \subseteq \mathbb{N}$ with

$$L \in \mathcal{L}(K\mathrm{ml}\tau\mathrm{0L}) \subseteq \mathcal{L}(\mathbb{N}\mathrm{ml}\tau\mathrm{0L}).$$

The reverse inclusion is true by Definition 2.31. ■

For fixed set K, the definition of the different types of KmlET0L languages immediately implies several trivial inclusion relations which are analogous to the relations (2.2) and (2.3) on page 16 in the case of non-limited ET0L languages. Using Lemma 2.35, the following theorem generalizes these relations for variable sets of permitted limits.

Theorem 2.38 For all non-empty sets K_1, K_2 with $K_1 \subseteq K_2 \subseteq \mathbb{N}$ and all $(\tau_1, \tau_2) \in \mathrm{CUBE}_{\mathrm{ET0L}}$ it holds that

$$\mathcal{L}(K_1\mathrm{ml}\tau_1\mathrm{0L}) \subseteq \mathcal{L}(K_2\mathrm{ml}\tau_2\mathrm{0L}). \blacksquare$$

For each five non-empty sets $K_1, \ldots, K_5$ with $K_1 \subseteq K_2 \subseteq K_3 \subseteq K_4 \subseteq K_5 \subseteq \mathbb{N}$ the inclusion relations of Theorem 2.38 form a 4-dimensional directed hyper cube-graph as represented in Figure 2.4 where the arrow relation $A \rightarrow B$ stands for $A \subseteq B$. Each such hyper cube-graph respectively consists of two connected 3-dimensional cube-graphs. The innermost cube-graph represents the inclusion relations between the eight families of non-extended mlT0L languages and the outermost cube-graph concerns the remaining eight families of mlET0L languages. For further details also see the description of Figure 2.1 on Page 17.

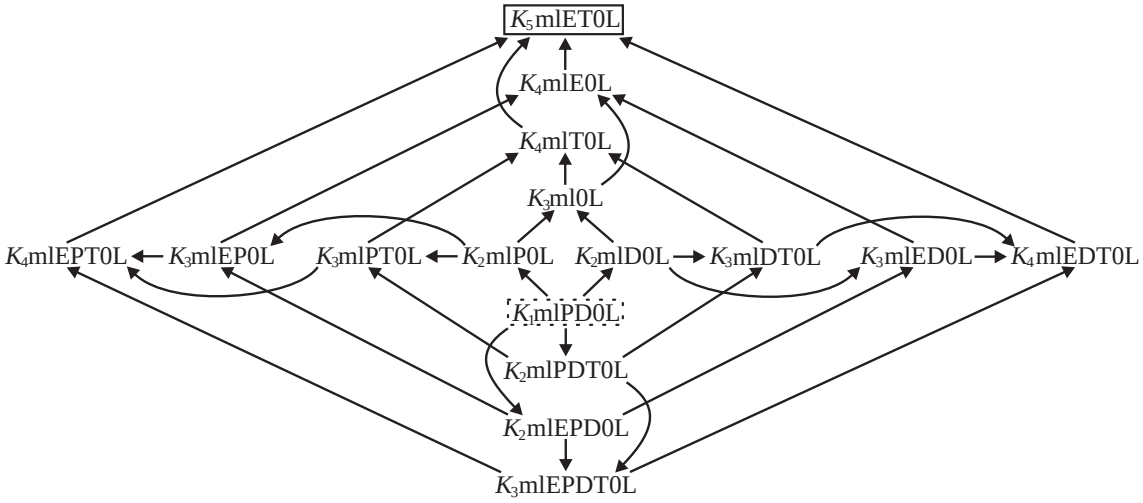

Figure 2.4. The *type cube* of the 16 families of mlET0L languages where $A \to B$ stands for $A \subseteq B$ and $K_1, \ldots, K_5$ are non-empty sets with $K_1 \subseteq K_2 \subseteq K_3 \subseteq K_4 \subseteq K_5 \subseteq \mathbb{N}$.

The following definition regarding word generating mechanisms does not only apply to multi-limited L systems, but also to all word generating systems based on context-free rewriting rules, and therefore, also to context-free grammars, non-limited L systems, as well as to k-limited L systems.

Definition 2.39 Given a binary yield relation $\Longrightarrow_G$ on Σ^* defined by system G which generates words over Σ by context-free rewriting rules. A sequence $D = (w_0, \ldots, w_n)$ of $n+1$ words over Σ with $n \in \mathbb{N}_0$ is called *derivation of length n according to system G* and denoted as

$$D : w_0 \Longrightarrow_G w_1 \Longrightarrow_G \ldots \Longrightarrow_G w_n$$

if $w_{i-1} \Longrightarrow_G w_i$ for all $i = 1, \ldots, n$. If only the initial and final word are of interest, then D also is written as $D : w_0 \Longrightarrow_G^n w_n$. Thus, $v \Longrightarrow_G^1 w$ means $v \Longrightarrow_G w$ and $v \Longrightarrow_G^0 w$ means $v = w$. A sequence $P = (a_0, \ldots, a_m)$ where a_i is an occurrence of a symbol of Σ within w_i, for all $i = 1, \ldots, m$ and $m \leq n$, is called *derivation path of length m according to derivation D of length n* and denoted as

$$P : a_0 \to_D a_1 \to_D \ldots \to_D a_m$$

if a_i is an occurrence within a subword x_i of w_i that arises from occurrence a_{i-1} according to derivation D, for all $i = 1, \ldots, m$. (Note: If occurrence a_{i-1} is not rewritten at all according to derivation D, then $x_i = a_i = a_{i-1}$.)

Occurrence b within w_0 is called *productive with respect to w_m according to D* if occurrence b is the beginning of a derivation path of length m according to derivation D. Otherwise, b is called *inproductive with respect to w_m according to D*. (Note: Each occurrence b within w_0 which is not rewritten according to D is productive with respect to w_m according to D.) ■

Example 2.40 Given the ml0L system $G = (\{a, b\}, h, a^2, \kappa)$ where $h(a) = \{\varepsilon, b\}$, $h(b) = \{\varepsilon\}$, $\kappa(a) = 1$ and $\kappa(b) = 2$. Then

$$D : w_0 = a^2 = c_1c_2 \Longrightarrow_G w_1 = ba = c_3c_4 \Longrightarrow_G w_2 = \varepsilon$$

is a derivation of length 2 according to system G where $c_1, \ldots, c_4$ denote certain occurrences within w_0 respectively w_1 and

(1) w_1 arises from w_0 by rewriting the occurrence c_1 of the symbol a by $c_3 = b \in h(a)$ and by non-rewriting the occurrence c_2 of the symbol a such that the occurrence c_4 arises from c_2, but not from c_1,
(2) w_2 arises from w_1 by rewriting the occurrence c_3 of the symbol b by $\varepsilon \in h(b)$ and the occurrence c_4 of the symbol a by $\varepsilon \in h(a)$.

Since $w_2 = \varepsilon$, no derivation path of length 2 exists at all according to D. Thus, all occurrences within w_0 are inproductive with respect to w_2 according to D. Furthermore, the sequence $P_1 = (c_2, c_4)$ is a derivation path of length 1 according to D and consequently, it can be written as

$$P_1 : c_2 \rightarrow_D c_4$$

By contrast, the sequence $P_2 = (c_2, c_3)$ is not a derivation path since the occurrence c_3 does not arise from the occurrence c_2 according to D. However, both occurrences, c_1 and c_2, are productive with respect to w_1 according to D. ■

Corollary 2.41 For a given derivation $D : w_0 \Longrightarrow_G w_1 \Longrightarrow_G \ldots \Longrightarrow_G w_n$ and each index $m \leq n$ the following holds:

(a) Each occurrence within w_0 which is productive with respect to w_m according to D also is productive with respect to w_i according to D for all $i < m$.
(b) Each occurrence within w_0 which is inproductive with respect to w_m according to D also is inproductive with respect to w_i according to D for all $i > m$. ■

Chapter 3

Graphical Modeling using L Systems

This chapter provides an intuitive approach to the different mechanisms of the L system variants considered in this work. Simultaneously, it outlines an application area of L systems outside formal language theory.

The biologist and mathematician Aristide Lindenmayer conceived L systems as a mathematical simulation of plant development. In this context, from its very beginning (see [17]), the graphical interpretation of the simulated organisms, i.e. of the generated strings, was of interest.

Strings generated by L systems may be interpreted geometrically in many different ways. In this chapter, the turtle interpretation of L systems is outlined which was introduced by Szilard and Quinton [34] and extended by Prusinkiewicz [19]. The latter constitutes the basis of the notation used in this chapter.

For the purpose of this thesis, the principal idea of the turtle interpretation of L systems shall be presented. Section 3.1 formalizes the notion of a two-dimensional turtle interpretation of arbitrary strings and provides examples of pictures generated by 0L systems under this interpretation. Section 3.2 extends the turtle interpretation to strings with brackets representing tree-like branching structures, and provides examples of tree-like pictures generated by non-limited, k-limited, and multi-limited 0L systems under the extended interpretation.

Further variants of turtle interpretation of strings, e.g. which use other geometrical

aspects like surfaces, colors, and line widths in two or three dimensions, or which create musical scores, can be found amongst others in [1], [19], and [23].

The examples presented in this chapter, regarding strings generated by 0L systems as well as regarding their turtle interpretations, were produced with the help of special C++ computer programs which were developed within the frame of this thesis. The respective source-code of a turtle interpreter as well as the source-code of free-programmable simulators for multi-limited, k-limited, and uniformly k-limited 0L systems can be found in the Appendix A of this work.

3.1 Turtle Interpretation of Strings

The principal idea of the turtle interpretation of L systems is as follows: After a string has been generated by an L system it is scanned sequentially from the left to the right and the consecutive symbols are interpreted as commands that maneuver a turtle in two dimensions. The turtle is represented by its state which consists of turtle position and orientation in the Cartesian coordinate system. Depending on the command, the turtle draws a line segment between the start position and the end position of the corresponding maneuver.

The following two definitions formalize the principal idea of the turtle interpretation of an arbitrary string.

Definition 3.1 A *turtle* is a quadruplet $T = (\Sigma, Z, d, \delta)$ where Σ is the alphabet of *commands* the turtle accepts, Z is the (infinite) set of *states*, d is the *step size*, and δ is the *angle increment* of the turtle. A state $z \in Z$ of the turtle is a triplet $z = (x, y, \alpha)$, where the Cartesian coordinates (x, y) represent the turtle's *position* within the plane, and the angle α, called the *heading*, is interpreted as the direction in which the turtle is facing.

Let $\{+, -\}$, Σ_F, and Σ_f be disjoint subsets of Σ. Then the turtle being in the state $z = (x, y, \alpha)$ responds to a command $c \in \Sigma$ as follows, otherwise the turtle preserves its current state:

$c \in \Sigma_F$	Move forward a step of length d. The state of the turtle changes to $z' = (x', y', \alpha)$, where $x' = x + d\cos\alpha$ and $y' = y + d\sin\alpha$. A line segment between the points (x, y) and (x', y') is drawn.
$c \in \Sigma_f$	Move forward a step of length d without drawing a line.
$c = +$	Turn right by the angle δ. The next state of the turtle is $z' = (x, y, \alpha + \delta)$.
$c = -$	Turn left by the angle δ. The next state of the turtle is $z' = (x, y, \alpha - \delta)$.

■

Definition 3.2 Given a turtle $T = (\Sigma, Z, d, \delta)$, an *initial state* $z_0 = (x_0, y_0, \alpha_0) \in Z$ of the turtle, and a word $w \in \Sigma^*$. Then the picture (the set of lines) drawn by the turtle beginning in the state z_0, responding to the word w, is called the *turtle interpretation of* w. ■

Subsequently, some well known examples of non-limited D0L systems and their turtle interpretations are presented.

Example 3.3 The curves included in Figure 3.1 belong to the class of *FASS* curves (an acronym for space-filling, self-avoiding, simple, and self-similar), which can be thought of as a finite, self-avoiding approximation of curves that pass through *all* points of the square. These special kind of FASS curves are named "hexagonal Gosper" curves (see [14]). The curves represent the turtle interpretation of the first five words generated by the non-limited D0L system $G = (\Sigma, h, \omega)$ with $\Sigma = \{L, R, +, -\}$, $\omega = L$, and the productions of h,

$$L \to L + R + +R - L - -LL - R+,$$
$$R \to -L + RR + +R + L - -L - R.$$

Thus, for example the first three words generated by G are

$$\begin{aligned}
w_0 &= L \\
w_1 &= L + R + +R - L - -LL - R+ \\
w_2 &= L + R + +R - L - -LL - R+ \quad + \\
&\quad -L + RR + +R + L - -L - R \quad ++ \\
&\quad -L + RR + +R + L - -L - R \quad - \\
&\quad L + R + +R - L - -LL - R+ \quad -- \\
&\quad L + R + +R - L - -LL - R+ \\
&\quad L + R + +R - L - -LL - R+ \quad - \\
&\quad -L + RR + +R + L - -L - R \quad +
\end{aligned}$$

The turtle interpretation of these words is generated by the turtle $T = (\Sigma, Z, d, \delta)$ with $\delta = 60°$, $\Sigma_F = \{L, R\}$, and $\Sigma_f = \emptyset$. In Figure 3.1, the initial state $z_0 = (x_0, y_0, \alpha_0)$ of the turtle is marked by an arrow leading into the depicted curves. Also the final state of the turtle is indicated by an arrow leading out of the depicted curves. ■

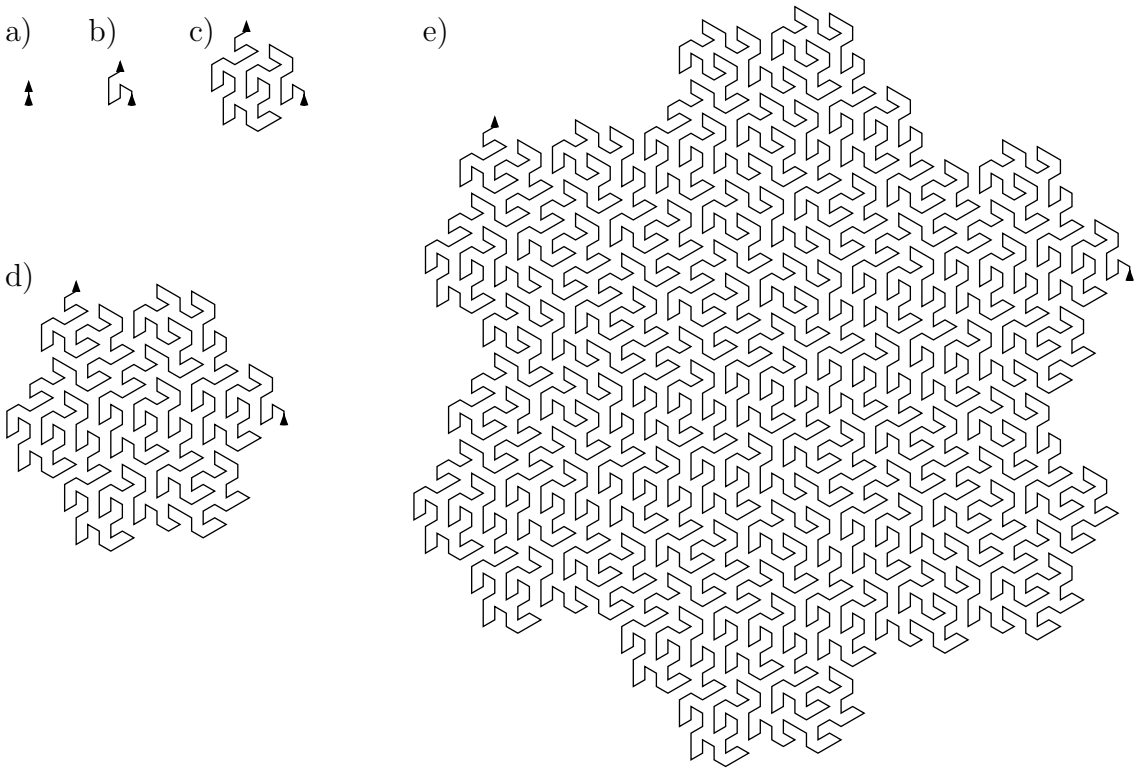

Figure 3.1. FASS curves named "hexagonal Gosper" curves, produced by the turtle interpretation with $\delta = 60°$ of the first five words a)–e) generated by the DOL system of Example 3.3 (see also Appendix A.1.2).

Example 3.4 The curves included in Figure 3.2 belong to the class of *Koch* curves (see [21]). The curves represent the turtle interpretation of the first three words generated by the non-limited D0L system $G = (\Sigma, h, \omega)$ with $\Sigma = \{F, f, +, -\}$, $\omega = F + F + F + F$, and the productions of h,

$$F \to F + f - FF + F + FF + Ff + FF - f + FF - F - FF - Ff - FFF,$$
$$f \to fffffff.$$

Thus, for example the first two words generated by G are

$$
\begin{aligned}
w_0 &= F + F + F + F \\
w_1 &= F + f - FF + F + FF + Ff + FF - f + FF - F - FF - Ff - FFF \; + \\
&\quad\; F + f - FF + F + FF + Ff + FF - f + FF - F - FF - Ff - FFF \; + \\
&\quad\; F + f - FF + F + FF + Ff + FF - f + FF - F - FF - Ff - FFF \; + \\
&\quad\; F + f - FF + F + FF + Ff + FF - f + FF - F - FF - Ff - FFF
\end{aligned}
$$

The turtle interpretation of these words is generated by the turtle $T = (\Sigma, Z, d, \delta)$ with $\delta = 90°$, $\Sigma_F = \{F\}$, and $\Sigma_f = \{f\}$. In Figure 3.2, the initial state $z_0 = (x_0, y_0, \alpha_0)$ of the turtle is marked by an arrow. ■

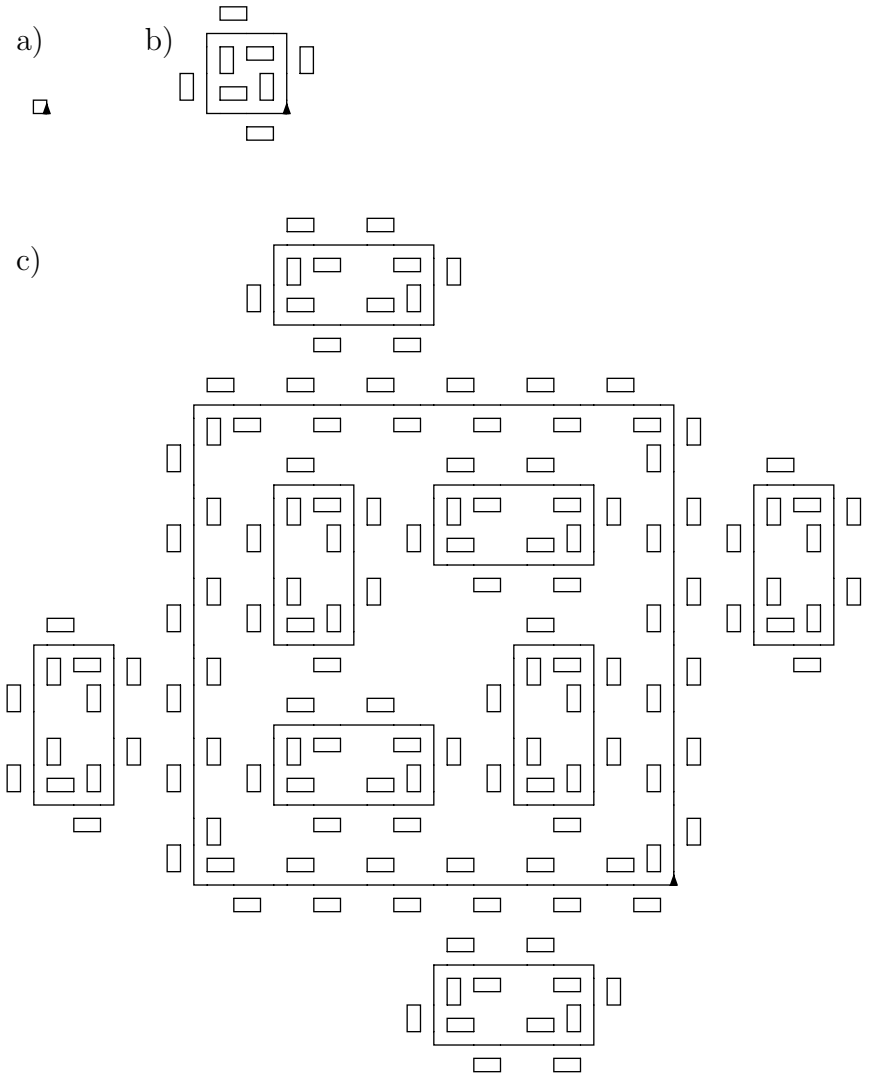

Figure 3.2. Quadratic Koch curves named "Combination of islands and lakes", produced by the turtle interpretation with $\delta = 90°$ of the first three words a)–c) generated by the DOL system of Example 3.4 (see also Appendix A.1.3).

3.2 Bracketed 0L Systems

In order to model the development of tree-like branching structures, Lindenmayer proposed so-called *bracketed* L systems in [17]. The alphabet of a bracketed L system contains the branch-delimiting bracket symbols '[' and ']'. The opening bracket '[' symbolizes the beginning of a branch and the corresponding closing bracket ']' marks the ending of the branch.

For a suitable turtle interpretation of this structure, a memory has to be added to the turtle which stores the current state at the branch base and recalls it at the end of the branch. Furthermore, the turtle needs the ability to jump back from the end to the beginning of the branch. These turtle extensions are formalized by the following two definitions.

Definition 3.5 A *bracketed turtle* is a quadruplet $T_b = (\Sigma \cup \{[,]\}, Z \times S, d, \delta)$ where $T = (\Sigma, Z, d, \delta)$ is a turtle, $S = \{(z_1, \ldots, z_n) \mid z_i \in Z,\ n \in \mathbb{N}_0\}$ is a *pushdown stack*, and '[' and ']' are additional turtle commands (*push* and *pop*). The *position* and the *heading* of T_b in the state $(z, s) \in Z \times S$ are defined as the position and the heading of T in the state z. The responds of T_b in the state (z, s) to a command $c \in \Sigma$ is defined as the responds of T in the state z, with the successor state (z', s) if z' is the successor state of T. To the additional commands '[' and ']' the bracketed turtle T_b in the state (z, s) responds as follows:

[Push the current state onto the pushdown stack S. The next state of T_b is $(z, (s, z))$.

] Pop a state from the pushdown stack S and make it the current state of the underlying turtle T. The next state of T_b is (z', s') if $s = (s', z')$ with $z' \in Z$. No line is drawn, although in general the position and orientation of the turtle are changed. ■

Definition 3.6 Given a bracketed turtle $T_b = (\Sigma \cup \{[,]\}, Z \times S, d, \delta)$, an *initial state* $(z_0, s_0) \in Z \times S$ of the turtle, and a word $w \in \Sigma^*$. Then the picture (the set of lines) drawn by the turtle beginning in the state (z_0, s_0), responding to the word w, is called the *bracketed turtle interpretation of* w. ■

The following examples present tree-like pictures generated by non-limited, k-limited, and multi-limited 0L systems under the extended interpretation. Simultaneously, they provide an intuitive approach to the different mechanisms of the L system variants considered in this work.

Example 3.7 The tree graphics included in Figure 3.3 represent the turtle interpretation of the first ten words generated by the non-limited bracketed D0L system $G = (\Sigma, h, \omega)$ with $\Sigma = \{a, b, c, l, r, w, x, +, -, [,]\}$, $\omega = a$, and the productions of h,

$$\begin{array}{llllll} a \to cwlb, & l \to [+a], & w \to x, & c \to c, & + \to +, & [\to [, \\ b \to cwra, & r \to [-b], & x \to wc, & & - \to -, &] \to]. \end{array}$$

In terms of plant development, the effects of the different productions of h can be described as follows. The cells c, w, and x, are lengthening cells which just enlarge the length of a piece of branch or stem. The cell l (r) generates a branch on the left (right) side of the current branch or stem. Finally, the cell a (b) generates a tree or subtree which begins with a piece of stem, cw, which just lengthens the stem in every derivation step, followed by the bud l (r) for a branch on the left (right) side and the bud b (a) for a subtree which first branches to the right (left). For example, the first six words generated by G are

$$\begin{aligned} w_0 &= a \\ w_1 &= cwlb \\ w_2 &= cx[+a]cwra \\ w_3 &= cwc[+cwlb]cx[-b]cwlb \\ w_4 &= cxc[+cx[+a]cwra]cwc[-cwra]cx[+a]cwra \\ w_5 &= cwcc[+cwc[+cwlb]cx[-b]cwlb]cxc[-cx[-b]cwlb]cwc[+cwlb]cx[-b]cwlb \end{aligned}$$

The turtle interpretation of these words is generated by the bracketed turtle $T_b = (\Sigma \cup \{[,]\}, Z \times S, d, \delta)$ with $\delta = 30°$, $\Sigma_F = \{a, b, c, l, r, w, x\}$, and $\Sigma_f = \emptyset$. In Figure 3.3, the initial state of the turtle is marked by a small arrow at the root of the tree, respectively. ■

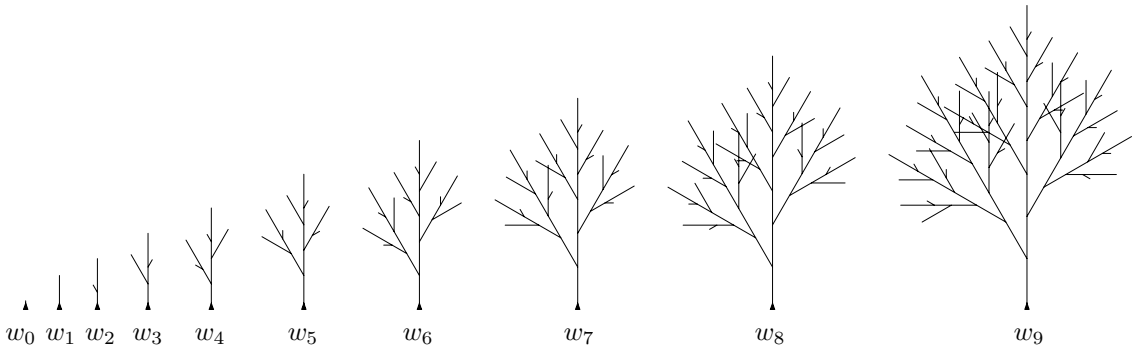

Figure 3.3. Tree-like branching structure, produced by the bracketed turtle interpretation with $\delta = 30°$ of the first ten words w_0–w_9 generated by the DOL system of Example 3.7 (see also Appendix A.1.1).

Example 3.8 Consider the non-limited D0L system $G = (\Sigma, h, \omega)$ of Example 3.7 and basing upon G, the k-limited D0L system $G_k = (\Sigma, h, \omega, 100)$ and the multi-limited D0L system $G_\kappa = (\Sigma, h, \omega, \kappa)$ with $\kappa(a) = 10$ and $\kappa(z) = 100$ for all $z \neq a$, $z \in \Sigma$. Using the same bracketed turtle T_b of Example 3.7, Figure 3.4 shows the bracketed turtle interpretation of three words which are derived from $\omega = a$ within 16 derivation steps according to G, G_k, and G_κ, respectively. Each time, the initial state of the turtle is marked by a tiny arrow at the root of the tree.

Figure 3.4 gives an impression regarding the influence of the different limitations. The left-most tree is generated by the non-limited D0L system and has a quite regular shape. The middle tree is generated by the k-limited D0L system. After an initial stage of exponential cell growth, all cell-types of the organism are growing linearly as in the presented stage. The right-most tree is generated by the multi-limited D0L system. Here, the resource for the cell-type a, representing the bud for a subtree which first branches to the left, is ten times smaller than the resources of the other cell-types. As a consequence of this, many branches of the organism turn clockwise. ■

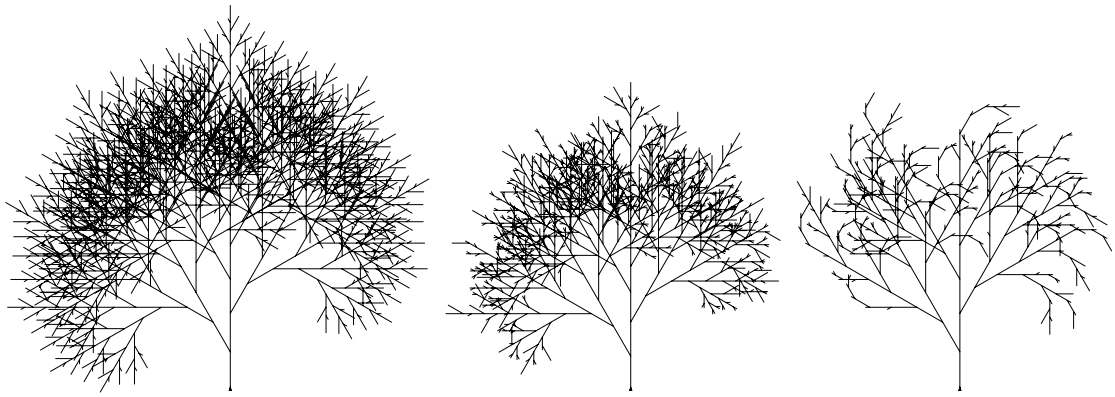

Figure 3.4. Bracketed turtle interpretation of three words generated within 16 derivation steps according to the non-limited, k-limited, and multi-limited D0L systems (from left to right) of Example 3.8 with angle increment $\delta = 30°$ (see also Appendix A.1.1).

Chapter 4

Multi-limited T0L Systems and Languages

In this chapter non-extended multi-limited 0L systems and languages are investigated. Regarding inclusion, mlT0L families are compared to each other (Section 4.1.1), to non-limited language families (Section 4.1.2), and to the Chomsky Hierarchy (Section 4.1.3). Closure properties of mlT0L families are investigated in Section 4.2.

4.1 Inclusion Relations

4.1.1 Comparison amongst the Families of mlT0L Languages

Lemma 4.1 Let $L_k = \{a^{2k}\} \cup \{w \in \{a,b\}^* \mid \#_a w = \#_b w = k\}$ for arbitrary $k \in \mathbb{N}$. Then $L_k \in \mathcal{L}(k\text{lPD0L}) \setminus \mathcal{L}(K\text{mlT0L})$ for each non-empty set $K \subseteq \mathbb{N} \setminus \{k\}$.

Proof: Let $k \in \mathbb{N}$. In order to prove that $L_k \in \mathcal{L}(k\text{lPD0L})$, consider the k-limited PD0L system

$$G_k = (\Sigma, h, \omega, k) \qquad \text{with } \Sigma = \{a,b\},\ h(a) = b,\ h(b) = a,\ \omega = a^{2k}.$$

On the one hand, $L_k \subseteq L(G_k)$ since $L_k = \{a^{2k}\} \cup h_k(a^{2k})$, i.e. all words of L_k can be generated within the first derivation step according to G_k. On the other hand,

the inclusion $L(G_k) \subseteq L_k$ can be proved by induction over the number $n \in \mathbb{N}_0$ of derivation steps according to G_k as follows.

The beginning $n = 0$ is trivial since $\omega = a^{2k} \in L_k$. For the conclusion, consider an arbitrary word w which can be derived within n steps according to G_k and thus, $w \in L_k$ by induction assumption. Then it has to be proved that $h_k(w) \subseteq L_k$. If $w = a^{2k}$, then $h_k(w) = L_k \setminus \{a^{2k}\} \subseteq L_k$. Otherwise, it remains the case that $w = x_1 \cdots x_{2k}$ with $x_1, \ldots, x_{2k} \in \{a, b\}$ and $\#_a w = \#_b w = k$. Then $h_k(w) = \overline{w} \in L_k$ with $\overline{w} = \overline{x}_1 \cdots \overline{x}_{2k}$ and $\overline{x}_i = a$ if $x_i = b$, and $\overline{x}_i = b$ if $x_i = a$, for $i = 1, \ldots, 2k$.

It remains to be proved that $L_k \notin \mathcal{L}(K\text{mlT0L})$ for all $K \subseteq \mathbb{N} \setminus \{k\}$. By Lemma 2.35, it suffices to consider the biggest family $\mathcal{L}(K_0\text{mlT0L})$ with $K_0 = \mathbb{N} \setminus \{k\}$, only. This proof is conducted by contradiction as follows.

Assume that L_k is generated by a K_0mlT0L system $G' = (\Sigma', H', \omega', \kappa')$. Since $L_k \subseteq \{a, b\}^*$, also $\Sigma' = \{a, b\}$ and $\kappa'(a) = k' \in K_0$ can be assumed. Then immediately it follows that $h'(a) \subseteq \{a, b\}$ for every $h' \in H'$, because otherwise, a word of length different from $2k$ could be derived from $a^{2k} \in L_k$, contradicting the assumption.

Thus, only two cases remain: either there exists a table $h'_0 \in H'$ with $b \in h'_0(a)$, or $h'(a) = \{a\}$ for every $h' \in H'$. Since $k' \neq 0$ and $k' \neq k$, the first case leads to the contradiction that either $a^{2k} \overset{h'_0}{\Longrightarrow} b^{k'} a^{2k-k'} \in L_k$ if $k' < 2k$, or $a^{2k} \overset{h'_0}{\Longrightarrow} b^{2k} \in L_k$ if $k' \geq 2k$. The second case implies that $\#_a w_1 = \#_a w_2$ for all $w_1, w_2 \in L_k$ contradicting that $a^{2k}, a^k b^k \in L_k$. ■

Theorem 4.2 For all non-empty sets $K_1, K_2 \subseteq \mathbb{N}$ and all $\tau_1, \tau_2 \in \text{TYPE}_{\text{T0L}}$ it holds that

$$\mathcal{L}(K_1\text{ml}\tau_1\text{0L}) \not\subseteq \mathcal{L}(K_2\text{ml}\tau_2\text{0L}) \qquad \text{if} \qquad K_1 \not\subseteq K_2.$$

Proof: Consider two non-empty sets $K_1, K_2 \subseteq \mathbb{N}$. If $K_1 \not\subseteq K_2$ there exists an integer $k \in K_1 \setminus K_2$. Using Lemma 4.1 this implies that

$$\mathcal{L}(k\text{lPD0L}) \setminus \mathcal{L}(K_2\text{mlT0L}) \neq \emptyset.$$

Since $\mathcal{L}(k\text{lPD0L}) \subseteq \mathcal{L}(K_1\text{ml}\tau_1\text{0L})$ and $\mathcal{L}(K_2\text{ml}\tau_2\text{0L}) \subseteq \mathcal{L}(K_2\text{mlT0L})$ for all $\tau_1, \tau_2 \in \text{TYPE}_{\text{T0L}}$ by Theorem 2.38, it follows that

$$\mathcal{L}(K_1\text{ml}\tau_1\text{0L}) \setminus \mathcal{L}(K_2\text{ml}\tau_2\text{0L}) \neq \emptyset$$

which completes the proof. ■

Theorem 4.2 immediately implies the following corollary which says that different sets $K \subseteq \mathbb{N}$ always lead to different families of languages generated by KmlT0L systems, independently from their types.

Corollary 4.3 For all non-empty sets $K_1, K_2 \subseteq \mathbb{N}$ and all $\tau_1, \tau_2 \in \mathrm{TYPE}_{\mathrm{T0L}}$ it holds that

$$\mathcal{L}(K_1\mathrm{ml}\tau_1\mathrm{0L}) \sim \mathcal{L}(K_2\mathrm{ml}\tau_2\mathrm{0L}) \qquad \text{if} \qquad K_1 \sim K_2$$

where the symbol '$\sim$' represents all binary relations of sets which can be built from the relation '$\not\subseteq$' by logical conjunction or disjunction, i.e. $\not\subseteq$, $\neq$, and 'is incomparable to'.

Proof: Consider arbitrary non-empty sets $K_1, K_2 \subseteq \mathbb{N}$ and arbitrary $\tau_1, \tau_2 \in \mathrm{TYPE}_{\mathrm{T0L}}$.

(a) If '$\sim$' is replaced by '$\not\subseteq$', this case is directly proved by Theorem 4.2.

(b) If $K_1 \neq K_2$, then $K_1 \not\subseteq K_2$ or $K_2 \not\subseteq K_1$. Thus, Theorem 4.2 implies that

$$\mathcal{L}(K_1\mathrm{ml}\tau_1\mathrm{0L}) \not\subseteq \mathcal{L}(K_2\mathrm{ml}\tau_2\mathrm{0L}) \text{ or } \mathcal{L}(K_2\mathrm{ml}\tau_2\mathrm{0L}) \not\subseteq \mathcal{L}(K_1\mathrm{ml}\tau_1\mathrm{0L}).$$

Consequently, also $\mathcal{L}(K_1\mathrm{ml}\tau_1\mathrm{0L}) \neq \mathcal{L}(K_2\mathrm{ml}\tau_2\mathrm{0L})$.

(c) If K_1 is incomparable to K_2, then $K_1 \not\subseteq K_2$ and $K_2 \not\subseteq K_1$. Thus, Theorem 4.2 implies that

$$\mathcal{L}(K_1\mathrm{ml}\tau_1\mathrm{0L}) \not\subseteq \mathcal{L}(K_2\mathrm{ml}\tau_2\mathrm{0L}) \text{ and } \mathcal{L}(K_2\mathrm{ml}\tau_2\mathrm{0L}) \not\subseteq \mathcal{L}(K_1\mathrm{ml}\tau_1\mathrm{0L}).$$

Consequently, also $\mathcal{L}(K_2\mathrm{ml}\tau_1\mathrm{0L})$ is incomparable to $\mathcal{L}(K_1\mathrm{ml}\tau_2\mathrm{0L})$. ■

If mlT0L families of the same type are compared to each other or in case of certain combinations of two mlT0L families of different types, Theorem 4.2 can be strengthened to the following theorem.

Theorem 4.4 For all non-empty sets $K_1, K_2 \subseteq \mathbb{N}$ and all pairs of types $(\tau_1, \tau_2) \in \mathrm{CUBE}_{\mathrm{T0L}}$ it holds that

$$\mathcal{L}(K_1\mathrm{ml}\tau_1\mathrm{0L}) \subseteq \mathcal{L}(K_2\mathrm{ml}\tau_2\mathrm{0L}) \qquad \text{if and only if} \qquad K_1 \subseteq K_2. \tag{4.1}$$

Proof: Consider arbitrary non-empty sets $K_1, K_2 \subseteq \mathbb{N}$ and an arbitrary pair of types $(\tau_1, \tau_2) \in \mathrm{CUBE}_{\mathrm{T0L}}$. If $K_1 \not\subseteq K_2$, then Theorem 4.2 implies that $\mathcal{L}(K_1\mathrm{ml}\tau_1\mathrm{0L}) \not\subseteq \mathcal{L}(K_2\mathrm{ml}\tau_2\mathrm{0L})$ since obviously $\tau_1, \tau_2 \in \mathrm{TYPE}_{\mathrm{T0L}}$.

Otherwise, if $K_1 \subseteq K_2$ Theorem 2.38 implies that $\mathcal{L}(K_1\text{ml}\tau_1\text{0L}) \subseteq \mathcal{L}(K_2\text{ml}\tau_2\text{0L})$ since obviously also $(\tau_1, \tau_2) \in \text{CUBE}_{\text{ET0L}}$. ■

For fixed type $\tau \in \text{TYPE}_{\text{T0L}}$ Theorem 4.4 means that the set K characterizes the family $\mathcal{L}(K\text{ml}\tau\text{0L})$. Together with Theorem 4.2, further statements regarding strict inclusions as well as regarding isomorphisms of mlT0L families are stated by the following Corollary 4.5.

Corollary 4.5 (a) For all non-empty sets $K_1, K_2 \subseteq \mathbb{N}$ and all $(\tau_1, \tau_2) \in \text{CUBE}_{\text{T0L}}$ it holds that

$$\mathcal{L}(K_1\text{ml}\tau_1\text{0L}) \subsetneqq \mathcal{L}(K_2\text{ml}\tau_2\text{0L}) \qquad \text{if} \qquad K_1 \subsetneqq K_2.$$

(b) For all non-empty sets $K_1, K_2 \subseteq \mathbb{N}$ and for fixed type $\tau \in \text{TYPE}_{\text{T0L}}$ it holds that

$$\mathcal{L}(K_1\text{ml}\tau\text{0L}) \sim \mathcal{L}(K_2\text{ml}\tau\text{0L}) \qquad \text{if and only if} \qquad K_1 \sim K_2$$

where the symbol '$\sim$' represents all binary relations of sets which can be built from the relation '$\subseteq$' by logical conjunction, disjunction, or negation, e.g. '$\subseteq$', '$\not\subseteq$', '$\subsetneqq$', '$=$', '$\neq$', and 'is incomparable to'.

(c) Consider the partial ordered sets $(\wp(\mathbb{N}) \setminus \emptyset, \subseteq)$ and $(S_\tau, \subseteq)$ with

$$S_\tau = \{\mathcal{L}(K\text{ml}\tau\text{0L}) \mid \emptyset \neq K \subseteq \mathbb{N}\}$$

for all $\tau \in \text{TYPE}_{\text{T0L}}$. Then all these sets are isomorphic to each other via the isomorphisms $I_\tau : (\wp(\mathbb{N}) \setminus \emptyset, \subseteq) \to (S_\tau, \subseteq)$ with $I_\tau(K) = \mathcal{L}(K\text{ml}\tau\text{0L})$ and $\tau \in \text{TYPE}_{\text{T0L}}$. Each isomorphism I_τ preserves the respective partial order '$\subseteq$'.

Proof: Consider arbitrary non-empty sets $K_1, K_2 \subseteq \mathbb{N}$.

(a) Let $K_1 \subsetneqq K_2$ and $(\tau_1, \tau_2) \in \text{CUBE}_{\text{T0L}}$. This implies, on the one hand, that $K_1 \subseteq K_2$ and consequently by Theorem 4.4 that

$$\mathcal{L}(K_1\text{ml}\tau_1\text{0L}) \subseteq \mathcal{L}(K_2\text{ml}\tau_2\text{0L}).$$

On the other hand, it also holds that $K_2 \not\subseteq K_1$ and $\tau_1, \tau_2 \in \text{TYPE}_{\text{T0L}}$. Thus,

$$\mathcal{L}(K_2\text{ml}\tau_2\text{0L}) \not\subseteq \mathcal{L}(K_1\text{ml}\tau_1\text{0L})$$

holds by Theorem 4.2 which proves item (a).

(b) If '$\sim$' is replaced by '$\subseteq$' or '$\not\subseteq$', these cases are directly proved by Theorem 4.4. If $K_1 \subsetneqq K_2$, then $K_1 \subseteq K_2$ and $K_2 \not\subseteq K_1$, and vice versa. Since $(\tau, \tau) \in \mathrm{CUBE}_{\mathrm{T0L}}$ for all $\tau \in \mathrm{TYPE}_{\mathrm{T0L}}$ Theorem 4.4 implies that this is equivalent to the conjunction

$$\mathcal{L}(K_1\mathrm{ml}\tau\mathrm{0L}) \subseteq \mathcal{L}(K_2\mathrm{ml}\tau\mathrm{0L}) \text{ and } \mathcal{L}(K_2\mathrm{ml}\tau\mathrm{0L}) \not\subseteq \mathcal{L}(K_1\mathrm{ml}\tau\mathrm{0L}).$$

Consequently, $K_1 \subsetneqq K_2$ if and only if $\mathcal{L}(K_1\mathrm{ml}\tau\mathrm{0L}) \subsetneqq \mathcal{L}(K_2\mathrm{ml}\tau\mathrm{0L})$.
The remaining cases, i.e. that '$\sim$' is replaced by '$=$', '$\neq$', respectively 'is incomparable to', can be proved analogously since

(i) $K_1 = K_2$ if and only if $K_1 \subseteq K_2$ and $K_2 \subseteq K_1$,
(ii) $K_1 \neq K_2$ if and only if $K_1 \not\subseteq K_2$ or $K_2 \not\subseteq K_1$,
(iii) K_1 is incomparable to K_2 if and only if $K_1 \not\subseteq K_2$ and $K_2 \not\subseteq K_1$.

(c) For each $\tau \in \mathrm{TYPE}_{\mathrm{T0L}}$ the mapping I_τ obviously is surjective by definition of S_τ. It also is injective since item (b) implies that $I_\tau(K_1) \neq I_\tau(K_2)$ if $K_1 \neq K_2$. Furthermore, I_τ preserves the partial order '$\subseteq$' since $I_\tau(K_1) \subseteq I_\tau(K_2)$ if and only if $K_1 \subseteq K_2$ by item (b). ■

Lemma 4.6 For all non-empty sets $K_1, K_2 \subseteq \mathbb{N}$ it holds that

$$\mathcal{L}(K_1\mathrm{mlD0L}) \not\subseteq \mathcal{L}(K_2\mathrm{mlPT0L}).$$

Proof: Consider arbitrary non-empty sets $K_1, K_2 \subseteq \mathbb{N}$ and the K_1mlD0L system $G = (\Sigma, h, \omega, \kappa)$ with $\Sigma = \{a\}$, $h(a) = \{\varepsilon, a\}$, $\omega = a$, and $\kappa(a) \in K_1$. Then G generates the language $L(G) = \{\varepsilon, a\}$.

But the language $L = \{\varepsilon, a\}$ cannot be generated by any K_2mlPT0L system $G' = (\Sigma', H', \omega', \kappa')$ which can be seen as follows. If $\omega' = \varepsilon$, then $L(G') = \{\varepsilon\} \neq L$. Otherwise, $\omega' = a$ and thus, $\varepsilon \notin L(G')$ since G' is propagating. ■

Lemma 4.7 For all non-empty sets $K_1, K_2 \subseteq \mathbb{N}$, except the case that $\{1\} = K_1 \subseteq K_2$, it holds that

$$\mathcal{L}(K_1\mathrm{mlP0L}) \not\subseteq \mathcal{L}(K_2\mathrm{mlDT0L}).$$

Proof: Consider arbitrary non-empty sets $K_1, K_2 \subseteq \mathbb{N}$. If $K_1 \not\subseteq K_2$, then $\mathcal{L}(K_1\mathrm{mlPD0L}) \not\subseteq \mathcal{L}(K_2\mathrm{mlDT0L})$ by Theorem 4.2. Since also $\mathcal{L}(K_1\mathrm{mlPD0L}) \subseteq \mathcal{L}(K_1\mathrm{mlP0L})$ by Theorem 4.4, the statement follows for $K_1 \not\subseteq K_2$.

Furthermore, for $K_1 \neq \{1\}$ consider the non-deterministic K_1mlP0L system $G = (\Sigma, h, \omega, \kappa)$ with $\Sigma = \{a, b\}$, $h(a) = \{a^2, ba\}$, $h(b) = \{b\}$, $\omega = a^2b^2$, and arbitrary limits $\kappa(a), \kappa(b) \geq 2$. Since G is propagating and both non-deterministic productions of G rewrite the symbol a by words of the same length, the five shortest words of $L(G)$ are a^2b^2, a^4b^2, a^2bab^2, baa^2b^2, and $babab^2$ which can be derived within the first derivation step according to G. Obviously,

$$L(G) \subseteq \{a, ba\}^+\{b^2\}$$

and thus, each word of $L(G)$ especially ends with ab^2 and contains exactly one occurrence of b^2.

But $L(G)$ cannot be generated by any K_2mlDT0L system with $\emptyset \neq K_2 \subseteq \mathbb{N}$. This is proved by contradiction: Assume that $L(G)$ is generated by a K_2mlDT0L system $G' = (\Sigma', H', \omega', \kappa')$. Then $\Sigma' = \Sigma = \{a, b\}$. At first it is shown by contradiction that the symbol b is not deleted according to G'.

Assume that $h'(b) = \varepsilon$ for some $h' \in H'$. Then there always exists a word $w \notin L(G)$ with $a^2b^2 \overset{h'}{\Longrightarrow}{}^* w$, a contradiction: If $\kappa'(b) > 1$ then b^2 is removed from a^2b^2. If $\kappa'(a) = 1$ then $w = va \notin L(G)$ is possible in the first step. If $\kappa'(a) > 1$ then $h'(a) = xab^2$ and consequently $w = xab^2xab^2 \notin L(G)$ is possible in the first step. If otherwise $\kappa'(b) = 1$, only one occurrence of b is removed from a^2b^2. If $\kappa'(a) = 1$ then $w = vab \notin L(G)$ is possible in the first step. If $\kappa'(a) > 1$ then $h'(a) = xab$ and consequently $a^2b^2 \overset{h'}{\Longrightarrow} xabxabb$ is possible in the first step. In the second step, deleting the last occurrence of b and rewriting at least both represented occurrences of a leads to $xabxabb \overset{h'}{\Longrightarrow} x'xabbx'xabb \notin L(G)$.

Thus, $h'(b) \in \{a, b\}^+$ for all $h' \in H'$. Furthermore, only $h'(b) = b$ is possible since otherwise, from a^2b^2 a word outside $L(G)$ could be derived which either does not end with b^2 or which contains more than one occurrence of b^2. By this result it follows that also the symbol a cannot be deleted by G' since otherwise, from a^2b^2 the words ab^2 or b^2 outside $L(G)$ could be derived if $\kappa'(a) = 1$ or $\kappa'(a) > 1$, respectively.

Consequently, G' is propagating with $h'(b) = b$ for all $h' \in H'$. But in this case the word $a^2bab^2 \in L(G)$ cannot be generated by G' which can be seen as follows. a^2bab^2 cannot be the axiom of G' and it only can be derived from one of the other four above mentioned shortest words of $L(G)$. Since $h'(b) = b$ for all $h' \in H'$, only $a^2b^2 \Longrightarrow a^2bab^2$ or $a^4b^2 \Longrightarrow a^2bab^2$ is possible. For symmetry reasons this implies that $\kappa'(a) = 1$ and thus, only the case that $a^2b^2 \Longrightarrow a^2bab^2$ remains. Consequently,

$h'(a) = a^2b$ or $h'(a) = aba$ for some $h' \in H'$. In both cases a word outside $L(G)$ can be derived from a^2b^2 which constitutes the final contradiction.

It only remains the case that $K_1 \subseteq K_2$ and $K_1 = \{1\}$. ■

It is of interest to note that the remaining case of $\{1\} = K_1 \subseteq K_2$ of Lemma 4.7 differs from all the other cases of Lemma 4.7, since $\mathcal{L}(\{1\}\text{mlP0L}) \subsetneqq \mathcal{L}(K_2\text{mlDT0L})$ for all $K_2 \supseteq \{1\}$ by Theorem 4.12 (see below).

Lemma 4.8 For all non-empty sets $K_1, K_2 \subseteq \mathbb{N}$ it holds that

$$\mathcal{L}(K_1\text{mlPDT0L}) \not\subseteq \mathcal{L}(K_2\text{ml0L}).$$

Proof: Consider arbitrary non-empty sets $K_1, K_2 \subseteq \mathbb{N}$ and the K_1mlPDT0L system $G = (\Sigma, H, \omega, \kappa)$ with $\Sigma = \{a, b\}$, $H = \{h_1, h_2\}$, $\omega = ab$, and $\kappa(a) = \kappa(b) \in K_1$, and $h_1(x) = x^3$, $h_2(x) = x^4$ for $x \in \Sigma$. Then $L(G) \subseteq \{a^nb^n \mid n \in \mathbb{N}\}$ and ab, a^3b^3, and a^4b^4 are the three shortest words of $L(G)$.

But $L(G)$ cannot be generated by any K_2ml0L system which is proved by contradiction: Assume that $L(G)$ is generated by a K_2ml0L system $G' = (\Sigma', h', \omega', \kappa')$. Then $\Sigma' = \Sigma = \{a, b\}$ and also $h'(a) \subseteq \{a\}^+$ and $h'(b) \subseteq \{b\}^+$ follows, since otherwise a word outside the set $\{a^nb^n \mid n \in \mathbb{N}\} \supseteq L(G')$ could be derived, a contradiction. Thus, G' is propagating and therefore, ω' is the shortest word of $L(G)$, i.e. $\omega' = ab$. Furthermore, since $a^3b^3 \in L(G)$ it follows that $a^3 \in h'(a)$ and $b^3 \in h'(b)$. Because G' is propagating the third-shortest word a^4b^4 of $L(G)$ only can be derived from a shorter word, ab or a^3b^3. Both cases lead to a contradiction as follows:

If $a^4b^4 \in h'(ab)$, then also $a^4 \in h'(a)$ and $b^4 \in h'(b)$. This leads to the contradiction $a^3b^4 \in h'(ab) \subseteq L(G)$.

Otherwise, if $a^4b^4 \in h'(a^3b^3)$, then also $a^2 \in h'(a)$ and $b^2 \in h'(b)$ since G' is propagating. This leads to the contradiction $a^3b^2 \in h'(ab) \subseteq L(G)$. ■

Theorem 4.9 For all non-empty sets $K_1, K_2 \subseteq \mathbb{N}$ and all $(\tau_1, \tau_2) \in \text{CUBE}_{\text{T0L}}$ with $\tau_1 \neq \tau_2$ it holds that

$$\mathcal{L}(K_1\text{ml}\tau_1\text{0L}) \subsetneqq \mathcal{L}(K_2\text{ml}\tau_2\text{0L}) \qquad \text{if} \qquad K_1 \subseteq K_2,$$

with the exception of

$$\mathcal{L}(\{1\}\text{mlPDT0L}) = \mathcal{L}(\{1\}\text{mlPT0L}) \text{ and } \mathcal{L}(\{1\}\text{mlDT0L}) = \mathcal{L}(\{1\}\text{mlT0L}).$$

Proof: The two exceptions hold because of $\mathcal{L}(1\text{lPDT0L}) = \mathcal{L}(1\text{lPT0L})$ and $\mathcal{L}(1\text{lDT0L}) = \mathcal{L}(1\text{lT0L})$ by [36], Theorem 3.3.

For the remaining cases consider two non-empty sets $K_1, K_2 \subseteq \mathbb{N}$ with $K_1 \subseteq K_2$ and a pair of type designators $(\tau_1, \tau_2) \in \text{CUBE}_{\text{T0L}}$ with $\tau_1 \neq \tau_2$. Note that for $K_1 \subsetneq K_2$ the statement has been proved already by Corollary 4.5 (a). Since the following proof does not take any advantage of this result, this case now will be proved for a second time, incidentally.

The simple inclusion $\mathcal{L}(K_1\text{ml}\tau_1\text{0L}) \subseteq \mathcal{L}(K_2\text{ml}\tau_2\text{0L})$ holds by Theorem 4.4. Therefore, it remains to prove that $\mathcal{L}(K_1\text{ml}\tau_1\text{0L}) \not\supseteq \mathcal{L}(K_2\text{ml}\tau_2\text{0L})$. This is done in the following for each considered pair (τ_1, τ_2) (as depicted in Figure 4.1).

(PD,P): Let $K_1 = K_2 = \{1\}$. Since $\mathcal{L}(1\text{lP0L}) \not\subseteq \mathcal{L}(1\text{lD0L})$ by [36], Lemma 3.3, and $\mathcal{L}(\{1\}\text{mlPD0L}) \subseteq \mathcal{L}(\{1\}\text{mlD0L})$ by Theorem 4.4, the statement follows.
It remains the case that $K_2 \neq \{1\}$. Since $\mathcal{L}(K_2\text{mlP0L}) \not\subseteq \mathcal{L}(K_2\text{mlDT0L})$ by Lemma 4.7 and $\mathcal{L}(K_1\text{mlPD0L}) \subseteq \mathcal{L}(K_2\text{mlDT0L})$ by Theorem 4.4, the statement follows also.

(PD,D): Since $\mathcal{L}(K_2\text{mlD0L}) \not\subseteq \mathcal{L}(K_1\text{mlPT0L})$ by Lemma 4.6 and $\mathcal{L}(K_1\text{mlPD0L}) \subseteq \mathcal{L}(K_1\text{mlPT0L})$ by Theorem 4.4, the statement follows.

(P,ε): Since $\mathcal{L}(K_2\text{mlD0L}) \not\subseteq \mathcal{L}(K_1\text{mlPT0L})$ by Lemma 4.6 and $\mathcal{L}(K_2\text{mlD0L}) \subseteq \mathcal{L}(K_2\text{ml0L})$ as well as $\mathcal{L}(K_1\text{mlP0L}) \subseteq \mathcal{L}(K_1\text{mlPT0L})$ by Theorem 4.4, the statement follows.

(D,ε): Let $K_1 = K_2 = \{1\}$. Since $\mathcal{L}(1\text{lP0L}) \not\subseteq \mathcal{L}(1\text{lD0L})$ by [36], Lemma 3.3, and $\mathcal{L}(\{1\}\text{mlP0L}) \subseteq \mathcal{L}(\{1\}\text{ml0L})$ by Theorem 4.4, the statement follows.
It remains the case that $K_2 \neq \{1\}$. Since $\mathcal{L}(K_2\text{mlP0L}) \not\subseteq \mathcal{L}(K_2\text{mlDT0L})$ by Lemma 4.7 and $\mathcal{L}(K_2\text{mlP0L}) \subseteq \mathcal{L}(K_2\text{ml0L})$ as well as $\mathcal{L}(K_1\text{mlD0L}) \subseteq \mathcal{L}(K_2\text{mlDT0L})$ by Theorem 4.4, the statement follows.

(PDT,PT): The case that $K_1 = K_2 = \{1\}$ already is covered by the first exception.
It remains the case that $K_2 \neq \{1\}$. Since $\mathcal{L}(K_2\text{mlP0L}) \not\subseteq \mathcal{L}(K_2\text{mlDT0L})$ by Lemma 4.7 and $\mathcal{L}(K_2\text{mlP0L}) \subseteq \mathcal{L}(K_2\text{mlPT0L})$ as well as $\mathcal{L}(K_1\text{mlPDT0L}) \subseteq \mathcal{L}(K_2\text{mlDT0L})$ by Theorem 4.4, the statement follows.

(PDT,DT): Since $\mathcal{L}(K_2\text{mlD0L}) \not\subseteq \mathcal{L}(K_1\text{mlPT0L})$ by Lemma 4.6 and $\mathcal{L}(K_2\text{mlD0L}) \subseteq \mathcal{L}(K_2\text{mlDT0L})$ as well as $\mathcal{L}(K_1\text{mlPDT0L}) \subseteq \mathcal{L}(K_1\text{mlPT0L})$ by Theorem 4.4, the statement follows.

(PT,T): Since $\mathcal{L}(K_2\text{mlD0L}) \not\subseteq \mathcal{L}(K_1\text{mlPT0L})$ by Lemma 4.6 and $\mathcal{L}(K_2\text{mlD0L}) \subseteq \mathcal{L}(K_2\text{mlT0L})$ by Theorem 4.4, the statement follows.

(DT,T): The case that $K_1 = K_2 = \{1\}$ already is covered by the first exception.

It remains the case that $K_2 \neq \{1\}$. Since $\mathcal{L}(K_2\text{mlP0L}) \not\subseteq \mathcal{L}(K_2\text{mlDT0L})$ by Lemma 4.7 and $\mathcal{L}(K_2\text{mlP0L}) \subseteq \mathcal{L}(K_2\text{mlT0L})$ as well as $\mathcal{L}(K_1\text{mlDT0L}) \subseteq \mathcal{L}(K_2\text{mlDT0L})$ by Theorem 4.4, the statement follows.

(PD,PDT): Since $\mathcal{L}(K_2\text{mlPDT0L}) \not\subseteq \mathcal{L}(K_1\text{ml0L})$ by Lemma 4.8 and $\mathcal{L}(K_1\text{mlPD0L}) \subseteq \mathcal{L}(K_1\text{ml0L})$ by Theorem 4.4, the statement follows.

(P,PT): Since $\mathcal{L}(K_2\text{mlPDT0L}) \not\subseteq \mathcal{L}(K_1\text{ml0L})$ by Lemma 4.8 and $\mathcal{L}(K_2\text{mlPDT0L}) \subseteq \mathcal{L}(K_2\text{mlPT0L})$ as well as $\mathcal{L}(K_1\text{mlP0L}) \subseteq \mathcal{L}(K_1\text{ml0L})$ by Theorem 4.4, the statement follows.

(D,DT): Since $\mathcal{L}(K_2\text{mlPDT0L}) \not\subseteq \mathcal{L}(K_1\text{ml0L})$ by Lemma 4.8 and $\mathcal{L}(K_2\text{mlPDT0L}) \subseteq \mathcal{L}(K_2\text{mlDT0L})$ as well as $\mathcal{L}(K_1\text{mlD0L}) \subseteq \mathcal{L}(K_1\text{ml0L})$ by Theorem 4.4, the statement follows.

(ε,T): Since $\mathcal{L}(K_2\text{mlPDT0L}) \not\subseteq \mathcal{L}(K_1\text{ml0L})$ by Lemma 4.8 and $\mathcal{L}(K_2\text{mlPDT0L}) \subseteq \mathcal{L}(K_2\text{mlT0L})$ by Theorem 4.4, the statement follows. ■

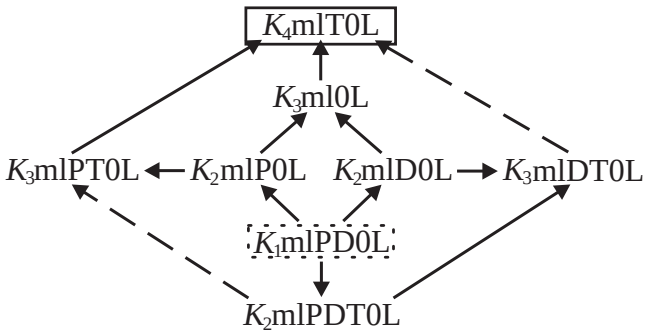

Figure 4.1. Inclusion relations of the 8 families of mlT0L languages where $A \to B$ stands for $A \subsetneqq B$ and $K_1, \ldots, K_4$ are non-empty sets with $K_1 \subseteq K_2 \subseteq K_3 \subseteq K_4 \subseteq \mathbb{N}$. The dashed lines mark the exceptions that $\mathcal{L}(\{1\}\text{mlPDT0L}) = \mathcal{L}(\{1\}\text{mlPT0L})$ and $\mathcal{L}(\{1\}\text{mlDT0L}) = \mathcal{L}(\{1\}\text{mlT0L})$.

Corollary 4.10 For each non-empty set $K \subseteq \mathbb{N}$ with $K \supsetneqq \{1\}$ it holds that

$$\mathcal{L}(K\text{mlPDT0L}) \supsetneqq \mathcal{L}(\{1\}\text{mlPT0L}) \text{ and } \mathcal{L}(K\text{mlDT0L}) \supsetneqq \mathcal{L}(\{1\}\text{mlT0L}).$$

Proof: Since $\mathcal{L}(\{1\}\text{mlPDT0L}) = \mathcal{L}(\{1\}\text{mlPT0L})$ by Theorem 4.9 (respectively by [36], Theorem 3.3), it follows by Theorem 4.4 that $\mathcal{L}(K\text{mlPDT0L}) \supseteq \mathcal{L}(\{1\}\text{mlPT0L})$ for each non-empty set $K \subseteq \mathbb{N}$ with $K \supsetneqq \{1\}$. Because $K \not\subseteq \{1\}$, Theorem 4.2 finally implies that $\mathcal{L}(K\text{mlPDT0L}) \supsetneqq \mathcal{L}(\{1\}\text{mlPT0L})$.

Analogous arguments lead to the fact that $\mathcal{L}(K\text{mlDT0L}) \supsetneqq \mathcal{L}(\{1\}\text{mlT0L})$ for each non-empty set $K \subseteq \mathbb{N}$ with $K \supsetneqq \{1\}$. ■

Remark 4.11 For all non-empty sets $K_1, K_2 \subseteq \mathbb{N}$ with $K_1 \supsetneqq K_2$ and all $(\tau_1, \tau_2) \in \mathrm{CUBE}_{\mathrm{T0L}}$ with $\tau_1 \neq \tau_2$, Theorem 4.2 guarantees that either

$$\mathcal{L}(K_1\mathrm{ml}\tau_1\mathrm{0L}) \supsetneqq \mathcal{L}(K_2\mathrm{ml}\tau_2\mathrm{0L}) \text{ or } \mathcal{L}(K_1\mathrm{ml}\tau_1\mathrm{0L}) \not\subseteq \not\supseteq \mathcal{L}(K_2\mathrm{ml}\tau_2\mathrm{0L}).$$

Except for the cases considered in Corollary 4.10, it remains open which of the two alternatives is true. ■

The above statements of Theorem 4.9, Corollary 4.10, and Remark 4.11 consider pairs of mlT0L families with type designators τ_1 and τ_2 where $(\tau_1, \tau_2) \in \mathrm{CUBE}_{\mathrm{T0L}}$ and $\tau_1 \neq \tau_2$. In Figure 4.1 these families are depicted as to be connected via a directed path. Now, the following Theorem 4.12 considers all remaining pairs of mlT0L families, i.e. which are not connected via a directed path.

Theorem 4.12 For all non-empty sets $K_1, K_2 \subseteq \mathbb{N}$ and all $\tau_1, \tau_2 \in \mathrm{TYPE}_{\mathrm{T0L}}$ where neither $(\tau_1, \tau_2) \in \mathrm{CUBE}_{\mathrm{T0L}}$ nor $(\tau_2, \tau_1) \in \mathrm{CUBE}_{\mathrm{T0L}}$ it holds that the families $\mathcal{L}(K_1\mathrm{ml}\tau_1\mathrm{0L})$ and $\mathcal{L}(K_2\mathrm{ml}\tau_2\mathrm{0L})$ are incomparable but not disjoint, with the exception of

$$\begin{aligned} \mathcal{L}(\{1\}\mathrm{mlP0L}) \subsetneqq \mathcal{L}(K\mathrm{mlDT0L}), \quad & \mathcal{L}(\{1\}\mathrm{mlP0L}) \subsetneqq \mathcal{L}(K\mathrm{mlPDT0L}), \\ \mathcal{L}(\{1\}\mathrm{ml0L}) \subsetneqq \mathcal{L}(K\mathrm{mlDT0L}), \quad & \mathcal{L}(\{1\}\mathrm{mlPT0L}) \subsetneqq \mathcal{L}(K\mathrm{mlDT0L}) \end{aligned}$$

for all $K \subseteq \mathbb{N}$ with $1 \in K$.

Proof: Let $K \subseteq \mathbb{N}$ with $1 \in K$. Then the four exceptions hold since Theorem 4.9 together with Corollary 4.10 imply that

$$\left.\begin{array}{l} \mathcal{L}(\{1\}\mathrm{ml0L}) \\ \mathcal{L}(\{1\}\mathrm{mlP0L}) \\ \mathcal{L}(\{1\}\mathrm{mlPT0L}) \end{array}\right\} \subsetneqq \mathcal{L}(\{1\}\mathrm{mlT0L}) \subseteq \mathcal{L}(K\mathrm{mlDT0L})$$

and also

$$\mathcal{L}(\{1\}\mathrm{mlP0L}) \subsetneqq \mathcal{L}(\{1\}\mathrm{mlPT0L}) \subseteq \mathcal{L}(K\mathrm{mlPDT0L}).$$

Furthermore, each two mlT0L families are not disjoint since they all obviously contain the language $\{a\}$.

For the remaining cases consider two non-empty sets $K_1, K_2 \subseteq \mathbb{N}$ and two type designators $\tau_1, \tau_2 \in \mathrm{TYPE}_{\mathrm{T0L}}$ where neither $(\tau_1, \tau_2) \in \mathrm{CUBE}_{\mathrm{T0L}}$ nor $(\tau_2, \tau_1) \in \mathrm{CUBE}_{\mathrm{T0L}}$. The incomparability of $\mathcal{L}(K_1\mathrm{ml}\tau_1\mathrm{0L})$ and $\mathcal{L}(K_2\mathrm{ml}\tau_2\mathrm{0L})$ is proved in the following for each considered pair (τ_1, τ_2).

(P,D): If $K_1 = \{1\}$, then $\mathcal{L}(K_1\text{mlP0L}) \not\subseteq \mathcal{L}(K_2\text{mlD0L})$ for each non-empty set $K_2 \subseteq \mathbb{N}$. This can be proved in the same way as it was used for the proof of $\mathcal{L}(\text{1lP0L}) \not\subseteq \mathcal{L}(\text{1lD0L})$ by [36], Lemma 3.3. The 1lP0L system $G = (\{a\}, \{a \to a^3, a \to a^4\}, a, 1)$ generates the language $L(G) = \{a, a^{3+n} \mid n \in \mathbb{N}_0\}$. Assume that $L(G)$ also is generated by a K_2mlD0L system $G' = (\Sigma, h, \omega, \kappa)$. Then obviously G' is propagating with $\Sigma = \{a\}$ and $\omega = a$. Furthermore, $a \Longrightarrow_{G'} a^3$ and consequently $h(a) = a^3$. This implies that $a^3 \Longrightarrow_{G'} a^{3+2m}$ where $m = \min\{3, \kappa(a)\} \geq 1$. But this contradicts the fact that only $a^3 \Longrightarrow_{G'} a^4$ is possible.

It remains the case that $K_1 \neq \{1\}$. Since $\mathcal{L}(K_1\text{mlP0L}) \not\subseteq \mathcal{L}(K_2\text{mlDT0L})$ by Lemma 4.7 and $\mathcal{L}(K_2\text{mlD0L}) \subseteq \mathcal{L}(K_2\text{mlDT0L})$ by Theorem 4.4 it holds that $\mathcal{L}(K_1\text{mlP0L}) \not\subseteq \mathcal{L}(K_2\text{mlD0L})$ for all non-empty $K_2 \subseteq \mathbb{N}$.

The opposite non-inclusion holds because $\mathcal{L}(K_1\text{mlPT0L}) \not\supseteq \mathcal{L}(K_2\text{mlD0L})$ by Lemma 4.6 and $\mathcal{L}(K_1\text{mlP0L}) \subseteq \mathcal{L}(K_1\text{mlPT0L})$ by Theorem 4.4.

(P,DT): The case that $K_1 = \{1\} \subseteq K_2$ already is covered by the exceptions. For the remaining cases it holds that $\mathcal{L}(K_1\text{mlP0L}) \not\subseteq \mathcal{L}(K_2\text{mlDT0L})$ by Lemma 4.7.

The opposite non-inclusion holds because $\mathcal{L}(K_1\text{ml0L}) \not\supseteq \mathcal{L}(K_2\text{mlPDT0L})$ by Lemma 4.8, and $\mathcal{L}(K_1\text{mlP0L}) \subseteq \mathcal{L}(K_1\text{ml0L})$ and $\mathcal{L}(K_2\text{mlPDT0L}) \subseteq \mathcal{L}(K_2\text{mlDT0L})$ by Theorem 4.4.

(P,PDT): The case that $K_1 = \{1\} \subseteq K_2$ already is covered by the exceptions. For the remaining cases it holds that $\mathcal{L}(K_1\text{mlP0L}) \not\subseteq \mathcal{L}(K_2\text{mlDT0L})$ by Lemma 4.7 and $\mathcal{L}(K_2\text{mlPDT0L}) \subseteq \mathcal{L}(K_2\text{mlDT0L})$ by Theorem 4.4 which implies $\mathcal{L}(K_1\text{mlP0L}) \not\subseteq \mathcal{L}(K_2\text{mlPDT0L})$.

The opposite non-inclusion holds because $\mathcal{L}(K_1\text{ml0L}) \not\supseteq \mathcal{L}(K_2\text{mlPDT0L})$ by Lemma 4.8 and $\mathcal{L}(K_1\text{mlP0L}) \subseteq \mathcal{L}(K_1\text{ml0L})$ by Theorem 4.4.

(D,PT): $\mathcal{L}(K_1\text{mlD0L}) \not\subseteq \mathcal{L}(K_2\text{mlPT0L})$ by Lemma 4.6.

The opposite non-inclusion holds because $\mathcal{L}(K_1\text{ml0L}) \not\supseteq \mathcal{L}(K_2\text{mlPDT0L})$ by Lemma 4.8, and $\mathcal{L}(K_1\text{mlD0L}) \subseteq \mathcal{L}(K_1\text{ml0L})$ and $\mathcal{L}(K_2\text{mlPDT0L}) \subseteq \mathcal{L}(K_2\text{mlPT0L})$ by Theorem 4.4.

(D,PDT): $\mathcal{L}(K_1\text{mlD0L}) \not\subseteq \mathcal{L}(K_2\text{mlPT0L})$ by Lemma 4.6 and $\mathcal{L}(K_2\text{mlPDT0L}) \subseteq \mathcal{L}(K_2\text{mlPT0L})$ by Theorem 4.4 which implies $\mathcal{L}(K_1\text{mlD0L}) \not\subseteq \mathcal{L}(K_2\text{mlPDT0L})$.

The opposite non-inclusion holds because $\mathcal{L}(K_1\text{ml0L}) \not\supseteq \mathcal{L}(K_2\text{mlPDT0L})$ by Lemma 4.8 and $\mathcal{L}(K_1\text{mlD0L}) \subseteq \mathcal{L}(K_1\text{ml0L})$ by Theorem 4.4.

(ε,PT): $\mathcal{L}(K_1\text{mlD0L}) \not\subseteq \mathcal{L}(K_2\text{mlPT0L})$ by Lemma 4.6 and $\mathcal{L}(K_1\text{mlD0L}) \subseteq \mathcal{L}(K_1\text{ml0L})$ by Theorem 4.4 which implies $\mathcal{L}(K_1\text{ml0L}) \not\subseteq \mathcal{L}(K_2\text{mlPT0L})$.

The opposite non-inclusion holds because $\mathcal{L}(K_1\text{ml0L}) \not\supseteq \mathcal{L}(K_2\text{mlPDT0L})$ by Lemma 4.8 and $\mathcal{L}(K_2\text{mlPDT0L}) \subseteq \mathcal{L}(K_2\text{mlPT0L})$ by Theorem 4.4.

(ε,DT): The case that $K_1 = \{1\} \subseteq K_2$ already is covered by the exceptions. For the remaining cases it holds that $\mathcal{L}(K_1\text{mlP0L}) \not\subseteq \mathcal{L}(K_2\text{mlDT0L})$ by Lemma 4.7 and $\mathcal{L}(K_1\text{mlP0L}) \subseteq \mathcal{L}(K_1\text{ml0L})$ by Theorem 4.4 which implies $\mathcal{L}(K_1\text{ml0L}) \not\subseteq \mathcal{L}(K_2\text{mlDT0L})$.

The opposite non-inclusion holds because $\mathcal{L}(K_1\text{ml0L}) \not\supseteq \mathcal{L}(K_2\text{mlPDT0L})$ by Lemma 4.8 and $\mathcal{L}(K_2\text{mlPDT0L}) \subseteq \mathcal{L}(K_2\text{mlDT0L})$ by Theorem 4.4.

(ε,PDT): $\mathcal{L}(K_1\text{mlD0L}) \not\subseteq \mathcal{L}(K_2\text{mlPT0L})$ by Lemma 4.6, and $\mathcal{L}(K_1\text{mlD0L}) \subseteq \mathcal{L}(K_1\text{ml0L})$ and $\mathcal{L}(K_2\text{mlPDT0L}) \subseteq \mathcal{L}(K_2\text{mlPT0L})$ by Theorem 4.4 which implies $\mathcal{L}(K_1\text{ml0L}) \not\subseteq \mathcal{L}(K_2\text{mlPDT0L})$.

The opposite non-inclusion holds because $\mathcal{L}(K_1\text{ml0L}) \not\supseteq \mathcal{L}(K_2\text{mlPDT0L})$ by Lemma 4.8.

(PT,DT): The case that $K_1 = \{1\} \subseteq K_2$ already is covered by the exceptions. For the remaining cases it holds that $\mathcal{L}(K_1\text{mlP0L}) \not\subseteq \mathcal{L}(K_2\text{mlDT0L})$ by Lemma 4.7 and $\mathcal{L}(K_1\text{mlP0L}) \subseteq \mathcal{L}(K_1\text{mlPT0L})$ by Theorem 4.4 which implies $\mathcal{L}(K_1\text{mlPT0L}) \not\subseteq \mathcal{L}(K_2\text{mlDT0L})$.

The opposite non-inclusion holds because $\mathcal{L}(K_1\text{mlPT0L}) \not\supseteq \mathcal{L}(K_2\text{mlD0L})$ by Lemma 4.6 and $\mathcal{L}(K_2\text{mlD0L}) \subseteq \mathcal{L}(K_2\text{mlDT0L})$ by Theorem 4.4. ■

Summary

The results of this section regarding inclusion relations amongst families of mlT0L languages are summarized in Table 4.1. As one can see, all relations regarding inclusions were completely resolved except those of Remark 4.11 where two alternatives remain.

No.	$(\tau_1, \tau_2) \in$ $\text{TYPE}^2_{\text{T0L}}$	$K_1 \sim K_2$ for non-empty $K_1, K_2 \subseteq \mathbb{N}$	$\mathcal{L}(K_1\text{ml}\tau_1\text{0L})$ $\sim \mathcal{L}(K_2\text{ml}\tau_2\text{0L})$	Reference
1.	$(\tau_1, \tau_2) \in$ CUBE_{T0L} except cases No. 2.-4.	$=$	$\subsetneq$	Theorem 4.9
		$\subsetneq$	$\subsetneq$	
		$\supsetneq$	$\supsetneq$ or $\not\subseteq\not\supseteq$	Remark 4.11
		$\not\subseteq\not\supseteq$	$\not\subseteq\not\supseteq$	Theorem 4.2
2.	$\tau_1 = \tau_2$	$=$	$=$	Corollary 4.5 (b)
		$\subsetneq$	$\subsetneq$	
		$\supsetneq$	$\supsetneq$	
		$\not\subseteq\not\supseteq$	$\not\subseteq\not\supseteq$	
3.	(PDT,PT)	$K_1 = K_2 = \{1\}$	$=$	Theorem 4.9 resp. [36], Theorem 3.3
		$K_1 \supsetneq K_2 = \{1\}$	$\supsetneq$	Corollary 4.10
4.	(DT,T)	$K_1 = K_2 = \{1\}$	$=$	Theorem 4.9 resp. [36], Theorem 3.3
		$K_1 \supsetneq K_2 = \{1\}$	$\supsetneq$	Corollary 4.10
5.	$(\tau_1, \tau_2), (\tau_1, \tau_2)$ $\notin \text{CUBE}_{\text{T0L}}$ except case No. 6.	$=$	$\not\subseteq\not\supseteq$	Theorem 4.12
		$\subsetneq$		
		$\supsetneq$		
		$\not\subseteq\not\supseteq$		
6.	(ε,DT)	$\{1\} = K_1 \subseteq K_2$	$\subsetneq$	Theorem 4.12
	(P,DT)			
	(PT,DT)			
	(P,PDT)			

Table 4.1. Summary of inclusion relations amongst families of mlT0L languages.

4.1.2 Comparison with the Families of T0L Languages

In this section it is shown that the families of multi-limited and the families of non-limited T0L languages do not include each other, in no direction.

Theorem 4.13 For every non-empty set $K \subseteq \mathbb{N}$ and every $\tau_1, \tau_2 \in \text{TYPE}_{\text{T0L}}$ the family $\mathcal{L}(K\text{ml}\tau_1\text{0L})$ is incomparable to each family $\mathcal{L}(\tau_2\text{0L})$, but they are not disjoint.

Proof: The considered families are not disjoint since obviously they all contain the finite language $\{a\}$. Furthermore, the language $L = \{a^{2^n} \mid n \in \mathbb{N}_0\} \in \mathcal{L}(\text{PD0L})$ by Example 2.18 (a), Page 16, but $L \notin \mathcal{L}(K\text{ml}\tau_1\text{0L})$ by Corollary 2.34, Page 25 for every non-empty set $K \subseteq \mathbb{N}$ and every $\tau_1 \in \text{TYPE}_{\text{T0L}}$. Since $\mathcal{L}(\text{PD0L}) \subseteq \mathcal{L}(\tau_2\text{0L})$ for all $\tau_2 \in \text{TYPE}_{\text{T0L}}$ by Lemma 2.20, Page 19, it holds that

$$\mathcal{L}(K\text{ml}\tau_1\text{0L}) \not\supseteq \mathcal{L}(\tau_2\text{0L})$$

for every non-empty set $K \subseteq \mathbb{N}$ and every $\tau_1, \tau_2 \in \text{TYPE}_{\text{T0L}}$. The opposite non-inclusions also hold because $\mathcal{L}(k\text{l}\tau_1\text{0L}) \not\subseteq \mathcal{L}(\tau_2\text{0L})$ for all $k \in \mathbb{N}$ and $\tau_1, \tau_2 \in \text{TYPE}_{\text{T0L}}$ by [36], Theorem 3.8, such that Theorem 4.4 completes the proof. ■

4.1.3 Comparison with the Chomsky Hierarchy

In this section the families of multi-limited T0L languages are compared to the families of the Chomsky Hierarchy regarding inclusion respectively non-inclusion.

Lemma 4.14 For every non-empty set $K \subseteq \mathbb{N}$, every $\tau \in \text{TYPE}_{\text{T0L}}$, and each family $\mathcal{L} \in \{\mathcal{L}(\text{fin}), \mathcal{L}(\text{reg}) \setminus \mathcal{L}(\text{fin}), \mathcal{L}(\text{cf}) \setminus \mathcal{L}(\text{reg}), \mathcal{L}(\text{cs}) \setminus \mathcal{L}(\text{cf})\}$ it holds that

$$\mathcal{L}(K\text{ml}\tau\text{0L}) \cap \mathcal{L} \neq \emptyset.$$

Proof: Since $\mathcal{L}(k\text{ml}\tau\text{0L}) \cap \mathcal{L} \neq \emptyset$ for all $k \in \mathbb{N}$ and $\tau \in \text{TYPE}_{\text{T0L}}$ by [36], Lemma 3.5, the statement immediately follows by Theorem 4.4. ■

Lemma 4.15 For every non-empty set $K \subseteq \mathbb{N}$ and every $\tau \in \text{TYPE}_{\text{T0L}}$ it holds that the finite language

$$\{a, a^2\} \notin \mathcal{L}(K\text{ml}\tau\text{0L}).$$

Proof: Assume that $\{a, a^2\} \in \mathcal{L}(K\text{mlT0L})$ for a non-empty set $K \subseteq \mathbb{N}$. Then $\{a, a^2\}$ is generated by a KmlT0L system $G = (\Sigma, H, \omega, \kappa)$. If $\omega = a$, then $a \Longrightarrow a^2$ and thus, $a^2 \Longrightarrow a^n$ with $n > 2$, a contradiction. Otherwise, $\omega = a^2$ which implies $a^2 \overset{h}{\Longrightarrow} a$ and consequently $\varepsilon \in h(a)$ for at least one table $h \in H$. Thus, $a \overset{h}{\Longrightarrow} \varepsilon \notin \{a, a^2\}$ which is the final contradiction to the assumption.

Now, the statement immediately follows by Theorem 4.4. ■

Lemma 4.16 For every non-empty set $K \subseteq \mathbb{N}$ and each family $\mathcal{L} \in \{\mathcal{L}(\text{fin}), \mathcal{L}(\text{reg}) \setminus \mathcal{L}(\text{fin}), \mathcal{L}(\text{cf}) \setminus \mathcal{L}(\text{reg}), \mathcal{L}(\text{cs}) \setminus \mathcal{L}(\text{cf})\}$ it holds that

$$\mathcal{L} \not\subseteq \mathcal{L}(K\text{mlT0L}).$$

Proof: In the following, let $K \subseteq \mathbb{N}$ be a non-empty set.

(a) $\{a, a^2\} \in \mathcal{L}(\text{fin}) \setminus \mathcal{L}(K\text{mlT0L})$ by Lemma 4.15.

(b) For $L = \{a, a^2\} \cup \{a^5\}^+ \in \mathcal{L}(\text{reg}) \setminus \mathcal{L}(\text{fin})$ assume that L is generated by a KmlT0L system $G = (\Sigma, H, \omega, \kappa)$. Then G is propagating, $\omega = a$, and $a \Longrightarrow a^2$. Thus, also $a^2 \Longrightarrow a^n$ with $n \in \{3, 4\}$. Consequently $a^n \notin L$ which contradicts the assumption.

(c) The language $L = \{a^n b^n \mid n \in \mathbb{N}\} \cup \{a^{n+1} b^n \mid n \in \mathbb{N}\}$ is context-free, but not regular (by the Lemma of *Bar-Hillel*). Assume that L is generated by a KmlT0L system $G = (\Sigma, H, \omega, \kappa)$. Then G is propagating, $\omega = ab$, and $h(x) \subseteq \{x\}^+$ for every table $h \in H$ and $x = a, b$. Since a^2b is the second shortest word in L, there exists a table $h \in H$ with $ab \overset{h}{\Longrightarrow} a^2b$. Consequently $a^2 \in h(a)$ and $b \in h(b)$. Thus, also $a^2b \overset{h}{\Longrightarrow} a^m b$ with $m > 2$. But this implies $a^m b \notin L$ which contradicts the assumption.

(d) Consider the language $L = \{a^{2^n} \mid n \in \mathbb{N}_0\} \in \mathcal{L}(\text{cs}) \setminus \mathcal{L}(\text{cf})$. By Corollary 2.34, Page 25, it holds that $L \notin \mathcal{L}(K\text{mlT0L})$ which completes the proof. ■

Theorem 4.17 For every non-empty set $K \subseteq \mathbb{N}$ and every $\tau \in \text{TYPE}_{\text{T0L}}$ there exist languages in each of the families $\mathcal{L}(\text{fin})$, $\mathcal{L}(\text{reg}) \setminus \mathcal{L}(\text{fin})$, $\mathcal{L}(\text{cf}) \setminus \mathcal{L}(\text{reg})$, and $\mathcal{L}(\text{cs}) \setminus \mathcal{L}(\text{cf})$ which belong to $\mathcal{L}(K\text{ml}\tau\text{0L})$ as well as languages which do not belong to $\mathcal{L}(K\text{ml}\tau\text{0L})$.

Proof: The first part of the statement just reflects the statement of Lemma 4.14. The second part follows by Lemma 4.16 together with Theorem 4.4. ■

Theorem 4.17 immediately implies the following Theorem 4.18. It is interesting to note that it is an unresolved question whether Theorem 4.18 also is valid for $\mathcal{L}(\text{cs})$.

Theorem 4.18 For every non-empty set $K \subseteq \mathbb{N}$ and every $\tau \in \text{TYPE}_{\text{T0L}}$ the family $\mathcal{L}(K\text{ml}\tau\text{0L})$ is incomparable to each family $\mathcal{L}(\text{fin})$, $\mathcal{L}(\text{reg})$, and $\mathcal{L}(\text{cf})$, but they are not disjoint. ■

Definition 4.19 Given an 0L system $G = (\Sigma, h, \omega)$, the corresponding *0S system* is defined as $G_{OS} = G$ with the *OS yield relation* $\Longrightarrow_{G_{OS}}$ on the set Σ^* where

$v \Longrightarrow_{G_{OS}} w$ holds if there exists a table $h \in H$ such that $w = w_1 a w_2$, $v = w_1 \beta w_2$, and $\beta \in h(a)$ for suitable $a \in \Sigma$ and $w_1, w_2 \in \Sigma^*$. The OS yield relation also is written as $v \overset{h}{\Longrightarrow}_{G_{OS}} w$ or, if it is unambiguous, simply $v \Longrightarrow w$. The transitive or reflexive transitive closure of $\Longrightarrow_{G_{OS}}$ is denoted by $\Longrightarrow^+_{G_{OS}}$ or $\Longrightarrow^*_{G_{OS}}$, respectively. Then the *OS language* $L(G_{OS})$ *generated by* G_{OS} is defined as

$$L(G_{OS}) = \{w \mid \omega \Longrightarrow^*_{G_{OS}} w\}.$$

The family of all OS languages is denoted by $\mathcal{L}(\mathrm{OS})$. ■

Since $\mathcal{L}(\mathrm{OS}) \subsetneqq \mathcal{L}(k\mathrm{l0L})$ for all $k \in \mathbb{N}$ by [36], Theorem 3.7, the following Theorem 4.20 holds by Theorem 4.4.

Theorem 4.20 For every non-empty set $K \subseteq \mathbb{N}$ it holds that $\mathcal{L}(\mathrm{OS}) \subsetneqq \mathcal{L}(K\mathrm{ml0L})$. ■

4.2 Closure Properties

Lemma 4.21 For $L = \{a^n b c^n \mid n \in \mathbb{N}\}$ it holds that $L^+ \notin \mathcal{L}(K\mathrm{mlT0L})$ for every non-empty set $K \subseteq \mathbb{N}$.

Proof: Let $K \subseteq \mathbb{N}$ be a non-empty set. Assume that L^+ is generated by a KmlT0L system $G = (\Sigma, H, \omega, \kappa)$ with $\Sigma = \{a, b, c\}$. Let $m = \max\{\kappa(a), \kappa(b), \kappa(c)\}$ and $h \in H$. At first it is shown that $h(a) = a$.

Consider the word $w = a^2bc^2(abc)^{m+1} \in L^+$. Then there exist at least two derivations $w \overset{h}{\Longrightarrow} w'$ and $w \overset{h}{\Longrightarrow} w''$ with $w' = v_a abc^2 av$ and $w'' = av_a bc^2 av$ where $v_a \in h(a)$ and v is a suitable word over Σ^* such that $w', w'' \in L^+$. On the one hand, because $w' \in L^+$ it follows that $v_a = xa$ with some $x \in L^*$. On the other hand, because also $w'' \in L^+$ it follows that $v_a = a$ or $v_a = a^{i-1}bc^i ya^2$ with some $i \in \mathbb{N}$ and $y \in L^*$. Consequently, there are only two cases. If $xa = a^{i-1}bc^i ya^2$, it follows that $x = a^{i-1}bc^i ya \in L^*$ which is a contradiction for all $i \in \mathbb{N}$ and $y \in L^*$. Therefore, only $v_a = a$ is possible and thus $h(a) = a$.

Analogous arguments imply that $h(c) = c$ using the words $w = (abc)^{m+1}a^2bc^2$, $w' = vca^2bcv_c$, and $w'' = vca^2bv_cc$.

Thus, if $\varepsilon \in h(b)$ for an arbitrary table $h \in H$, the derivation $abc \overset{h}{\Longrightarrow} ac \notin L^+$ leads to a contradiction. Therefore, $\varepsilon \notin h(b)$ for all $h \in H$ and G is propagating with $\omega = abc$. Consider the three shortest words of L^+, i.e. abc, a^2bc^2, and $abcabc$. Then the longest of these words, i.e. $abcabc$, can be derived from abc or a^2bc^2, only. The latter case is not possible because of $h(a) = \{a\}$ and $h(c) = \{c\}$ for all $h \in H$. Consequently, there exists a table $h \in H$ such that $abc \overset{h}{\Longrightarrow} abcabc$ and thus, $bcab \in h(b)$. But this finally leads to the contradiction that $a^3bc^3 \overset{h}{\Longrightarrow} a^3bcabc^3 \notin L^+$. ■

Lemma 4.22 $L = \{ab, ba, b^3\} \notin \mathcal{L}(K\text{mlT0L})$ for every non-empty set $K \subseteq \mathbb{N}$.

Proof: Let $K \subseteq \mathbb{N}$ be a non-empty set. Assume that $L = \{ab, ba, b^3\}$ is generated by a KmlT0L system $G = (\Sigma, H, \omega, \kappa)$ with $\Sigma = \{a, b\}$. If there exists a table $h \in H$ with $\varepsilon \in h(b)$, then $b^3 \overset{h}{\Longrightarrow}{}^* \varepsilon \notin L$ leads to a contradiction. If there exists a table $h \in H$ and a word $w \in h(b)$ of length > 1, then a word of length > 3 can be derived from b^3 which also is a contradiction. Thus, only $h(b) \subseteq \{a, b\}$ is possible. If $a \in h(b)$, then it holds that $b^3 \overset{h}{\Longrightarrow} v$ where $v \in \{abb, aab, aaa\}$, a contradiction for each case. Consequently, $h(b) = b$ for all $h \in H$. Since $h(b^3) = b^3$ it follows that either $\omega = ab$ or $\omega = ba$. But both cases lead to a contradiction because neither $ab \Longrightarrow^* ba$ nor $ba \Longrightarrow^* ab$ is possible. ■

Lemma 4.23 $L = \{a^3, a^2b, ba, b^2\} \notin \mathcal{L}(K\text{mlT0L})$ for every non-empty set $K \subseteq \mathbb{N}$.

Proof: Let $K \subseteq \mathbb{N}$ be a non-empty set. Assume that $L = \{a^3, a^2b, ba, b^2\}$ is generated by a KmlT0L system $G = (\Sigma, H, \omega, \kappa)$ with $\Sigma = \{a, b\}$. First it is shown that $h(a) \subseteq \{a, \varepsilon\}$ and $h(b) \subseteq \{b, \varepsilon\}$ for arbitrary table $h \in H$.

If there exists a word $w \in h(a)$ with $w \neq a$ and $w \neq \varepsilon$, then it holds that either $a^3 \overset{h}{\Longrightarrow} w^3 \notin L$, if $\kappa(a) \geq 3$, or $a^3 \overset{h}{\Longrightarrow} w^2a \notin L$, if $\kappa(a) = 2$, or $a^3 \overset{h}{\Longrightarrow} awa \notin L$, if $\kappa(a) = 1$. All three alternatives lead to a contradiction.

If there exists a word $w \in h(b)$ with $w \neq b$ and $w \neq \varepsilon$, then it holds that either $b^2 \overset{h}{\Longrightarrow} w^2 \notin L$, if $\kappa(b) \geq 2$, or $b^2 \overset{h}{\Longrightarrow} bw$, if $\kappa(b) = 1$. The first alternative leads to a contradiction. For the second alternative, it follows that $bw \in L$ and thus, $w = a$. But this leads to the contradiction that $b^2 \overset{h}{\Longrightarrow} wb = ab \notin L$.

Consequently, $h(a) \subseteq \{a, \varepsilon\}$ and $h(b) \subseteq \{b, \varepsilon\}$ for all $h \in H$. This implies that no derivation according to G can enlarge the length of a processed word. Therefore,

either $\omega = a^3$ or $\omega = a^2b$. The first case leads to the contradiction that $L(G) \subseteq \{a\}^*$. The second case leads to the contradiction that $L(G) \subseteq \{a\}^*\{b\}^*$. ■

Theorem 4.24 For every non-empty set $K \subseteq \mathbb{N}$ and every $\tau \in \text{TYPE}_{\text{T0L}}$, the family $\mathcal{L}(K\text{ml}\tau\text{0L})$ is an anti-AFL and additionally not closed with respect to concatenation.

Proof: According to Definition 2.13, Page 13, it suffices to prove that each of the considered families is not closed with respect to any of the six types of operations: (a) union, (b) ε-free iteration, (c) intersection with regular languages, (d) ε-free homomorphism, (e) inverse homomorphism, and (f) concatenation. The proof is conducted by the construction of counterexamples for each operation. By Theorem 4.4 it suffices to prove that every family $\mathcal{L}(K\text{mlPD0L})$, for arbitrary non-empty set $K \subseteq \mathbb{N}$, contains languages which operation result is not an mlT0L language, at all.

(a) Obviously $\{a\}, \{a^2\} \in \mathcal{L}(K\text{mlPD0L})$, but by Lemma 4.15 it holds that

$$\{a\} \cup \{a^2\} = \{a, a^2\} \notin \mathcal{L}(\text{mlT0L}).$$

(b) The KmlPD0L system $G = (\{a, b, c\}, h, abc, \kappa)$ with $h(a) = a$, $h(b) = abc$, and $h(c) = c$, generates $L = \{a^n bc^n \mid n \in \mathbb{N}\}$. But $L^+ \notin \mathcal{L}(\text{mlT0L})$ by Lemma 4.21.

(c) The language generated by the KmlPD0L system $G = (\{a\}, h, a, \kappa)$ with $h(a) = a^2$ obviously contains the regular set $R = \{a, a^2\}$. Therefore, by Lemma 4.15 it holds that

$$L(G) \cap R = \{a, a^2\} \notin \mathcal{L}(\text{mlT0L}).$$

(d) The KmlPD0L system $G = (\{b, c\}, h, b, \kappa)$ with $h(b) = c$ and $h(c) = b$, generates $L = \{b, c\} \in \mathcal{L}(K_1\text{mlPD0L})$. Consider the ε-free homomorphism g with $g(b) = a$ and $g(c) = a^2$. Then by Lemma 4.15 it holds that

$$g(L) = \{a, a^2\} \notin \mathcal{L}(\text{mlT0L}).$$

(e) Obviously $L = \{a^3\} \in \mathcal{L}(K\text{mlPD0L})$. Consider the homomorphism h with $h(a) = a^2$ and $h(b) = a$. Then by Lemma 4.22 it holds that

$$h^{-1}(L) = \{ab, ba, b^3\} \notin \mathcal{L}(\text{mlT0L}).$$

(f) The KmlPD0L system $G_n = (\{a, b\}, h, b, \kappa)$ with $h(a) = a$ and $h(b) = a^n$, generates $L_n = \{a^n, b\} \in \mathcal{L}(K_1\text{mlPD0L})$ for $n \in \mathbb{N}$. But Lemma 4.23 implies that

$$L_2 \cdot L_1 = \{a^3, a^2b, ba, b^2\} \notin \mathcal{L}(\text{mlT0L}). \blacksquare$$

Definition 4.25 Given an alphabet Σ, the mapping $mir : \Sigma^* \to \Sigma^*$, $mir(a_1 \cdots a_n) = a_n \cdots a_1$ for $a_1, \ldots, a_n \in \Sigma$ and $n \in \mathbb{N}_0$, is called *mirror image*. The mirror image is extended to languages $L \subseteq \Sigma^*$ by setting

$$mir(L) = \{mir(w) \mid w \in L\}. \blacksquare$$

Theorem 4.26 For every non-empty set $K \subseteq \mathbb{N}$ and every $\tau \in \text{TYPE}_{\text{T0L}}$, the family $\mathcal{L}(K\text{ml}\tau\text{0L})$ is closed with respect to mirror image.

Proof: Let L be generated by a Kmlτ0L system $G = (\Sigma, H, \omega, \kappa)$ with arbitrary non-empty set $K \subseteq \mathbb{N}$ and $\tau \in \text{TYPE}_{\text{T0L}}$. Then $mir(L)$ obviously is generated by the Kmlτ0L system $G' = (\Sigma, H', mir(\omega), \kappa)$ where $H' = \{h' \mid h \in H\}$ with $h'(a) = mir(h(a))$ for all $h' \in H'$ and $a \in \Sigma$. $\blacksquare$

Chapter 5

Multi-limited ET0L Systems and Languages

In this chapter extended multi-limited 0L systems and languages are investigated. Section 5.1 presents a normal form of mlET0L systems. Regarding inclusion, mlET0L families are compared to each other (Section 5.2.1), to non-limited language families (Section 5.2.2), and to the Chomsky Hierarchy (Section 5.2.3). Closure properties of mlET0L families are investigated in Section 5.3.

5.1 Normal Form of mlET0L Systems

The following Theorem 5.1 describes a *normal form* of mlET0L systems which means that each mlET0L language can be generated by an mlET0L system of this form. Beside others, this fact is used to prove Theorem 5.18 of Section 5.2.1.

Theorem 5.1 (Normal Form Theorem) For arbitrary non-empty sets $K \subseteq \mathbb{N}$, each language $L \in \mathcal{L}(K\text{mlET0L})$ can be generated by a KmlET0L system $G = (\Sigma, H, \omega, \Delta, \kappa)$ of the normal form which fulfills each of the following properties:

(a) $\omega \in \Sigma \setminus \Delta$,
(b) there exists a *failure symbol* $F \in \Sigma \setminus \Delta$ with $h(F) = \{F\}$ for all $h \in H$,
(c) there exists a *terminal table* $h_T \in H$ with $h_T(x) \in \Delta \cup \{F\}$ for all $x \in \Sigma$ and $h_T(a) = \{a\}$ for all $a \in \Delta$,

(d) all the remaining tables $h \in H \setminus \{h_T\}$ fulfill that $h(x) = \{F\}$ if $x \in \Delta$, and $h(x) \subseteq (\Sigma \setminus \Delta)^*$, otherwise.

Furthermore, if $L \in \mathcal{L}(K\text{mlEPT0L})$, $\mathcal{L}(K\text{mlEDT0L})$, or $\mathcal{L}(K\text{mlEPDT0L})$, the normal form G can be chosen as KmlEPT0L, KmlEDT0L, or KmlEPDT0L system, respectively.

Proof: Let L be generated by an arbitrary KmlET0L system $G_0 = (\Sigma_0, H_0, \omega_0, \Delta, \kappa_0)$ with a non-empty set $K \subseteq \mathbb{N}$. Then the normal form $G = (\Sigma, H, \omega, \Delta, \kappa)$ with the properties (a)–(d) is constructed as follows.

For each $w \in \Sigma_0^*$ let w' denote the word that results from w by rewriting each occurrence of a terminal symbol $a \in \Delta$ by a new non-terminal symbol $a' \in \Sigma \setminus \Delta$. Furthermore, let $S' = \{w' \mid w \in S\}$ for every set $S \subseteq \Sigma_0^*$. Define the parameters of the normal form G as

$$\Sigma = \Sigma_0 \cup \Delta' \cup \{\omega, F\} \qquad \text{and} \qquad H = \{h \mid h_0 \in H_0\} \cup \{h_I, h_T\}$$

with the new symbols $\omega, F \notin \Sigma_0 \cup \Delta'$ and

$$h_I(x) = \begin{cases} \{\omega_0'\} & \text{if } x = \omega, \\ \{F\} & \text{if } x \in \Delta, \end{cases}$$

$$h_T(x) = \begin{cases} \{a\} & \text{if } x = a' \in \Delta', \\ \{F\} & \text{if } x \in \Sigma \setminus (\Delta' \cup \Delta), \end{cases}$$

$$h(x) = \begin{cases} (h_0(x))' & \text{if } x \in \Sigma_0 \setminus \Delta, \\ (h_0(a))' & \text{if } x = a' \in \Delta', \\ \{F\} & \text{if } x \in \Delta \end{cases}$$

for every $h \in H$ with $h_0 \in H_0$. Set $h(x) = \{x\}$ for every value $h(x)$ which remained unspecified in the above specification for $h \in H$ and $x \in \Sigma$. Finally define

$$\kappa(x) = \begin{cases} \kappa_0(x) & \text{if } x \in \Sigma_0, \\ \kappa_0(a) & \text{if } x = a' \in \Delta', \\ k_0 & \text{otherwise} \end{cases}$$

for every $x \in \Sigma$, where k_0 is an arbitrary but fixed element of K (e.g. $k_0 = \min K$). By this construction, it is obvious that the system G is a KmlEPT0L, KmlEDT0L, or KmlEPDT0L system if the system G_0 is a KmlEPT0L, KmlEDT0L, or

KmlEPDT0L system, respectively. Furthermore, $L(G_0) = L(G)$ since every derivation sequence according to G_0 with

$$\omega_0 \overset{h_0^{(1)}}{\Longrightarrow} w_1 \overset{h_0^{(2)}}{\Longrightarrow} w_2 \overset{h_0^{(3)}}{\Longrightarrow} \ldots \overset{h_0^{(r)}}{\Longrightarrow} w_r \in \Delta^* \tag{5.1}$$

where $h_0^{(i)} \in H_0$ and $w_i \in \Sigma_0^*$ for $i = 1, \ldots, r$, is equivalent to the corresponding derivation sequence according to G

$$\omega \overset{h_I}{\Longrightarrow} \omega_0' \overset{h^{(1)}}{\Longrightarrow} w_1' \overset{h^{(2)}}{\Longrightarrow} w_2' \overset{h^{(3)}}{\Longrightarrow} \ldots \overset{h^{(r)}}{\Longrightarrow} w_r' \overset{h_T}{\Longrightarrow}{}^* w_r \tag{5.2}$$

which can be seen as follows. Obviously, (5.1) implies (5.2). For the proof of the reversal, consider an arbitrary derivation sequence according to G

$$\omega \overset{h^{(0)}}{\Longrightarrow}_G v_0 \overset{h^{(1)}}{\Longrightarrow}_G v_1 \overset{h^{(2)}}{\Longrightarrow}_G \ldots \overset{h^{(n)}}{\Longrightarrow}_G v_n \in \Delta^*. \tag{5.3}$$

It is proved that on the one hand, (5.3) coincides with (5.2), except for derivation steps which are of no effect with respect to the derived terminal word v_n, and that on the other hand, (5.1) holds. Without introducing the failure symbol F and without applying ineffective derivation steps the tables of H can be applied in the following order, only:

1. h_I: Starting at the initial symbol ω, without introducing the failure symbol F and without leaving the input unchanged, the table h_I can be applied, only. Since no table $h \in H$ can introduce ω, it follows that $h^{(0)} = h_I$, $v_0 = \omega_0'$, and $h^{(i)} \neq h_I$ for all $i > 0$.
2. $h \in H$ with $h_0 \in H_0$: Since $h(a) = F$ for all terminal symbols $a \in \Delta$, the table h cannot be applied after the terminal table h_T has been applied. Furthermore, the definition of h and κ immediately implies that

$$w' \overset{h}{\Longrightarrow}_G v' \qquad \text{if and only if} \qquad w \overset{h_0}{\Longrightarrow}_{G_0} v$$

   for all words $v, w \in \Sigma_0^*$, because $h(x') = (h_0(x))'$ and $\kappa(x') = \kappa_0(x)$ for all symbols $x \in \Sigma_0$. Thus, since $v_0 = \omega_0' \in \Sigma_0'^*$ by step 1, there exists an index $r \leq n$ for (5.3) such that $h^{(i)} \in H$ with $h_0^{(i)} \in H_0$ as well as $v_i = w_i'$ with $w_i \in \Sigma_0^*$, $w_0 = \omega_0$, and $w_{i-1} \overset{h_0^{(i)}}{\Longrightarrow}_{G_0} w_i$, for $i = 1, \ldots, r$. The value of r is determined in step 3.
3. h_T: If the word $v_r = w_r' = \varepsilon$ has been derived in step 2, then $r = n$ neglecting possible ineffective derivations steps, and both derivations, (5.2) as well as (5.1), hold with $w_r = \varepsilon$. Otherwise, the table h_T has to be applied for at least

one time since no other table of H can introduce terminal symbols. Without introducing the failure symbol F, only the table h_T can be applied to a word $w \in (\Delta' \cup \Delta)^*$. Since no other table $h \in H$ can be applied to terminal symbols, it follows that the table h_T has to be applied to a word $w'_r \in \Delta'^+$ for the first time. This determines the value of r. Because h_T introduces at least one terminal symbol, it follows that $h^{(i)} = h_T$ and $v'_i = w'_r$ for all $i = r+1, \ldots, n$ where at least $v_n = w_r \in \Delta^*$. ■

Example 5.2 Consider the $\{1,2\}$mlPD0L system $G_0 = (\{a,b\}, h_0, ab, \kappa_0)$ with $h_0(a) = ab$, $h_0(b) = abb$, and $\kappa_0(a) = 2$, $\kappa_0(b) = 1$. Then the normal form corresponding to G_0 is represented by the $\{1,2\}$mlEPD0L system

$$G = (\{S, a', b', a, b\}, \{h_I, h_T, h\}, S, \{a, b\}, \kappa)$$

with

$$\begin{array}{cccccc} h_I(S) = a'b', & h_I(a') = a', & h_I(b') = b', & h_I(a) = F, & h_I(b) = F, & h_I(F) = F, \\ h(S) = S, & h(a') = a'b', & h(b') = a'b'b', & h(a) = F, & h(b) = F, & h(F) = F, \\ h_T(S) = F, & h_T(a') = a, & h_T(b') = b, & h_T(a) = a, & h_T(b) = b, & h_T(F) = F, \end{array}$$

as well as $\kappa(a) = \kappa(a') = 2$ and $\kappa(b) = \kappa(b') = \kappa(S) = 1$. For a demonstration that the normal form G is equivalent to the system G_0, consider for example the derivation of length 2 according to G_0, where always the leftmost occurrences are rewritten, i.e.

$$ab \overset{h_0}{\Longrightarrow} ab\,abb \overset{h_0}{\Longrightarrow} ab\,abb\,ab\,bb.$$

Then the corresponding derivation according to the normal form G is

$$S \overset{h_I}{\Longrightarrow} a'b' \overset{h}{\Longrightarrow} a'b'\,a'b'b' \overset{h}{\Longrightarrow} a'b'\,a'b'b'\,a'b'\,b'b' \overset{h_T}{\Longrightarrow} ab\,ab'b'\,a'b'\,b'b' \overset{h_T}{\Longrightarrow}{}^* ab\,abb\,ab\,bb. \blacksquare$$

5.2 Inclusion Relations

In this section extended multi-limited 0L systems and languages are investigated regarding inclusion. The mlET0L families are compared to each other (Section 5.2.1), to non-limited language families (Section 5.2.2), and to the Chomsky Hierarchy (Section 5.2.3).

5.2.1 Comparison amongst the Families of mlET0L Languages

The following theorem describes some necessary properties of mlE0L languages. Therefore, it can be used in order to prove that some languages are not a member of $\mathcal{L}$(mlE0L). The proof of this theorem bases on the proof of a similar result for k-limited 0L systems (see [40]).

Theorem 5.3 (Weak Iteration Theorem) Let $L = L(G)$ be generated by a KmlE0L system $G = (\Sigma, h, \omega, \Delta, \kappa)$ with a non-empty set $K \subseteq \mathbb{N}$. Let $L' \subseteq L$ be an (infinite) sublanguage over a subalphabet $\Delta' = \{a_1, \ldots, a_n\} \subseteq \Delta$, $n \in \mathbb{N}$, such that for all $r \in \mathbb{N}$ there exists a word $w \in L'$ with $\#_a w > r$ for all $a \in \Delta'$. Then at least one of the following properties, (a) or (b), and also at least one of the following properties, (a') or (b'), hold.

(a) For all $m \in \mathbb{N}$ there exist $m+1$ words $w_0, \ldots, w_m \in L \cap \Delta'^*$ and a non-zero constant $c \in \mathbb{Z}$ such that either
  (1) $|w_{i+1}| = |w_i| + c$ for all $i = 0, \ldots, m-1$, or
  (2) $|w_{i+1}| = |w_i|$ and $\#_a w_{i+1} = \#_a w_i + c$ for all $i = 0, \ldots, m-1$ and a suitable symbol $a \in \Delta'$.

(b) There exist words $t_{a_j} \in h(a_j) \cap \Delta'^*$ for $j = 1, \ldots, n$ such that $t_{a_i} = \varepsilon$ for at least one index $i \in \{1, \ldots, n\}$ and $\#_{a_j}(t_{a_1}^{\kappa(a_1)} \cdots t_{a_n}^{\kappa(a_n)}) = \kappa(a_j)$ for all $j = 1, \ldots, n$.

(a') For all $m \in \mathbb{N}$ there exist $m+1$ words $w_0, \ldots, w_m \in L \cap \Delta'^*$ and a non-zero constant $c \in \mathbb{Z}$ such that $|w_{i+1}| = |w_i| + c$ for all $i = 0, \ldots, m-1$.

(b') $\varepsilon \in h(a)$ for at least one symbol $a \in \Delta'$.

Proof: Consider the languages L and $L' \subseteq L$ and a system G with the above assumed properties. Let $k = \max\{\kappa(a) \mid a \in \Sigma\}$ and $s = \max\{|w| \mid w \in h(a),\ a \in \Sigma\}$. Since L' and thus L is infinite, it follows that $s > 1$. Let $m \in \mathbb{N}$. Then the assumptions imply that there exists a word $w_0 \in L'$ such that

$$\#_a w_0 > \sigma = k \cdot m \cdot |\Sigma| \cdot s^{|\Sigma|} \cdot |\omega| \qquad \text{for all } a \in \Delta'.$$

Since in each derivation step, at most $k \cdot |\Sigma| \cdot s < \sigma$ occurrences of each symbol $a \in \Sigma$ can be generated according to G, there exists a predecessor $\tilde{w}$ of w_0 with $\omega \Longrightarrow^* \tilde{w} \Longrightarrow w_0$ and $\#_a \tilde{w} > 0$ for all $a \in \Delta'$. Consequently, there exists a word $t_a \in h(a) \cap \Delta'^*$ for all $a \in \Delta'$. Furthermore, since also $\sigma > k \cdot n \cdot m$, the table h can

be applied to w_0 for at least m times, each time rewriting $\kappa(a)$ occurrences of each $a \in \Delta'$ by t_a. These applications of h lead to a derivation

$$w_0 \Longrightarrow^* w_1 \Longrightarrow^* \ldots \Longrightarrow^* w_m$$

where for $i = 0, \ldots, m$

$$|w_i| = |w_0| + i \cdot \underbrace{\sum_{a \in \Delta'} \kappa(a) \cdot (|t_a| - 1)}_{=: c} \qquad \text{and} \qquad w_i \in L \cap \Delta'^*.$$

If $c \neq 0$, the (identical) properties (a)(1) and (a') are fulfilled. Otherwise, if $c = 0$, there are only two cases: either $|t_a| = 0$ for at least one $a \in \Delta'$ or $|t_a| = 1$ for all $a \in \Delta'$. In the first case, the property (b') is fulfilled. Also the property (b) holds, if additionally $\#_{a_j}(t_{a_1}^{\kappa(a_1)} \cdots t_{a_n}^{\kappa(a_n)}) = \kappa(a_j)$ for all $j = 1, \ldots, n$. Otherwise, there exists a non-zero constant $c' \in \mathbb{Z}$ such that $\#_{a_{j_0}}(t_{a_1}^{\kappa(a_1)} \cdots t_{a_n}^{\kappa(a_n)}) = \kappa(a_{j_0}) + c'$ for at least one index $j_0 = 1, \ldots, n$. Consequently, $\#_{a_{j_0}} w_{i+1} = \#_{a_{j_0}} w_i + c'$ for all $i = 0, \ldots, m-1$, which implies the property (a)(2).

For the remaining case $|t_a| = 1$ for all $a \in \Delta'$, consider the derivation

$$D : \omega \Longrightarrow^* w_0.$$

Since $|w_0| > \sigma > |\omega| \cdot s^{|\Sigma|-1}$, the derivation D contains at least one *cyclic occurrence* $x \in \Sigma$ which means that

$$D : \omega \Longrightarrow^* u_1'' x u_2'' \Longrightarrow^* u_1' v_1' x v_2' u_2' \Longrightarrow^* u_1 v_1 v_0 v_2 u_2 = w_0$$

with $v_1 v_2 \neq \varepsilon$ and

$$x \Longrightarrow_{(1)}^* v_1' x v_2', \quad x \Longrightarrow_{(1)}^* v_0, \quad v_j' \Longrightarrow_{(1)}^* v_j, \quad u_j' \Longrightarrow_{(1)}^* u_j$$

for $j = 1, 2$, where $\Longrightarrow_{(1)}^*$ symbolizes the respective sequential rewritings according to D. (Note that without any cyclic occurrence, only words of length less than $|\omega| \cdot s^{|\Sigma|-1}$ can be derived from ω.) Now, by using this cyclic occurrence x, a new sequence of $m + 1$ words $\tilde{w_0}, \ldots, \tilde{w_m} \in L \cap \Delta'^*$ is constructed which fulfills the properties (a)(1) and (a'). Let $\tilde{w_0} = w_0$. Consider for $i = 1, \ldots, m$ the derivation

$$D_i : \omega \Longrightarrow^* u_1' v_1' x v_2' u_2' \Longrightarrow^* u_1^{(i)} v_{1,0}^{(i)} \cdots v_{1,i}^{(i)} v_0^{(i)} v_{2,i}^{(i)} \cdots v_{2,0}^{(i)} u_2^{(i)} = \tilde{w_i}$$

with

$$x \Longrightarrow_{(i)}^* v_0^{(i)}, \quad v_j' \Longrightarrow_{(i)}^* v_{j,t}^{(i)}, \quad u_j' \Longrightarrow_{(i)}^* u_j^{(i)}$$

for $j = 1, 2$ and $t = 0, \ldots, i$, where in each derivation $x \Longrightarrow^*_{(i)} v_0^{(i)}$ the respective sequential rewritings according to the derivation $x \Longrightarrow^*_{(1)} v_0$ are used with the only exception that each occurrence of a symbol $a \in \Delta'$ within v_0 ($v_0 \in \Delta'^*$) is rewritten by $t_a \in \Delta'$. Analogously, $v'_j \Longrightarrow^*_{(i)} v^{(i)}_{j,t}$ corresponds to $v'_j \Longrightarrow^*_{(1)} v_j$, and $u'_j \Longrightarrow^*_{(i)} u^{(i)}_j$ corresponds to $u'_j \Longrightarrow^*_{(1)} u_j$, for $j = 1, 2$ and $t = 0, \ldots, i$. Thus,

$$|v_0^{(i)}| = |v_0|, \quad |v^{(i)}_{j,t}| = |v_j|, \quad |u^{(i)}_j| = |u_j|$$

for $j = 1, 2$ and $t = 0, \ldots, i$. This finally implies that, for $i = 0, \ldots, m$,

$$|\tilde{w}_i| = |\tilde{w}_0| + i \cdot c' \qquad \text{with} \qquad c' = |v_1 v_2| > 0. \blacksquare$$

Example 5.4 The following languages are not members of $\mathcal{L}$(mlE0L) since they fulfill all assumptions of Theorem 5.3, but none of the implications (a) or (b) (also none of the implications (a') or (b')):

(a) $\{a^{2^n} \mid n \in \mathbb{N}_0\}$,
(b) $\{a^p \mid p \text{ is a prime number}\}$,
(c) $\{a^{n^2} \mid n \in \mathbb{N}\}$. ■

In the sequel, inclusion relations between families of mlET0L languages are investigated which types are of $\text{CUBE}_{\text{ET0L}} \setminus \text{CUBE}_{\text{T0L}}$. The following result of Lemma 5.5 is proved basing on [36], Example 4.2(b).

Lemma 5.5 For all non-empty sets $K_1, K_2 \subseteq \mathbb{N}$ it holds that

$$\mathcal{L}(K_1\text{mlEPD0L}) \not\subseteq \mathcal{L}(K_2\text{mlT0L}).$$

Proof: Let $K_1, K_2 \subseteq \mathbb{N}$ be arbitrary non-empty sets. Consider the K_1mlEPD0L system $G = (\Sigma, h, a, \Delta, \kappa)$ with $\Sigma = \Delta \cup \{F\}$, $\Delta = \{a, b\}$, an arbitrary mapping $\kappa : \Sigma \to K_1$, and

$$h(a) = a^2 b, \qquad h(b) = F, \qquad h(F) = F.$$

Then G generates the language $L = \{a, a^2 b\}$. Assume that L also is generated by any K_2mlT0L system $G' = (\Sigma, H, \omega', \kappa')$ with $\Sigma = \Delta$ and $\omega' \in L$. If $\omega' = a$, there exists a table $h \in H$ with $a^2 b \in h(a)$. Hence, from $a^2 b$ a word with at least three occurrences of the symbol a can be derived which does not belong to L. Otherwise, the word a can be derived from $\omega' = a^2 b$. Consequently, there exists a table $h' \in H$ with $\varepsilon \in h'(a)$ and thus, $\varepsilon \notin L$ can be derived from $a \in L$. ■

With the help of Lemma 5.5 the following Theorem can be deduced.

Theorem 5.6 For all $\tau_1 \in \text{TYPE}_{\text{ET0L}} \setminus \text{TYPE}_{\text{T0L}}$, $\tau_2 \in \text{TYPE}_{\text{T0L}}$, and all non-empty sets $K_1, K_2 \subseteq \mathbb{N}$ it holds that

$$\mathcal{L}(K_1\text{ml}\tau_1\text{0L}) \not\subseteq \mathcal{L}(K_2\text{ml}\tau_2\text{0L}).$$

Proof: The assertion immediately follows by Lemma 5.5 and Theorem 2.38, page 26, because the latter implies that, on the one hand, $\mathcal{L}(K_1\text{mlEPD0L}) \subseteq \mathcal{L}(K_1\text{ml}\tau_1\text{0L})$, for all $\tau_1 \in \text{TYPE}_{\text{ET0L}} \setminus \text{TYPE}_{\text{T0L}}$, and on the other hand, $\mathcal{L}(K_2\text{ml}\tau_2\text{0L}) \subseteq \mathcal{L}(K_2\text{mlT0L})$, for all $\tau_2 \in \text{TYPE}_{\text{T0L}}$. ■

It remains open whether the corresponding reverse result of Lemma 5.5, regarding mlPDT0L languages and mlE0L languages, also is true. It is supposed that the answer is 'yes' (see Conjecture 5.11 below). Nevertheless, a weaker result can be proved easily.

Lemma 5.7 For all non-empty sets $K_1, K_2 \subseteq \mathbb{N}$ it holds that

$$\mathcal{L}(K_1\text{mlEPDT0L}) \not\subseteq \mathcal{L}(K_2\text{mlE0L}).$$

Proof: Consider the language $L = \{a^{2^n} \mid n \in \mathbb{N}_0\}$. Then $L \in \mathcal{L}(K_1\text{mlEPDT0L})$ for every non-empty set $K_1 \subseteq \mathbb{N}$ by Example 2.25 (b), page 21. But $L \notin \mathcal{L}(K_2\text{mlE0L})$ for every non-empty set $K_2 \subseteq \mathbb{N}$ by Example 5.4 (a), page 63. ■

Theorem 5.8 For all $\tau_1 \in \{\text{ET,EPT,EDT,EPDT}\}$, $\tau_2 \in \text{TYPE}_{\text{E0L}}$, and all non-empty sets $K_1, K_2 \subseteq \mathbb{N}$ it holds that

$$\mathcal{L}(K_1\text{ml}\tau_1\text{0L}) \not\subseteq \mathcal{L}(K_2\text{ml}\tau_2\text{0L}).$$

Proof: The assertion immediately follows by Lemma 5.7 together with Theorem 2.38, page 26, because the latter implies that, on the one hand, $\mathcal{L}(K_1\text{mlEPDT0L}) \subseteq \mathcal{L}(K_1\text{ml}\tau_1\text{0L})$, for all $\tau_1 \in \{\text{ET,EPT,EDT,EPDT}\}$, and on the other hand, $\mathcal{L}(K_2\text{ml}\tau_2\text{0L}) \subseteq \mathcal{L}(K_2\text{mlE0L})$, for all $\tau_2 \in \text{TYPE}_{\text{E0L}}$. ■

Now, Theorem 5.8 and Theorem 5.6 can be used to prove the following two theorems regarding strict inclusions.

Theorem 5.9 For all $(\tau_1, \tau_2) \in \text{CUBE}_{\text{E0L}}$ and all non-empty sets $K_1, K_2 \subseteq \mathbb{N}$ with $K_1 \subseteq K_2$ it holds that

$$\mathcal{L}(K_1\text{ml}\tau_1\text{0L}) \subsetneqq \mathcal{L}(K_2\text{ml}\tau_2\text{T0L}).$$

Proof: If $(\tau_1, \tau_2) \in \mathrm{CUBE}_{\mathrm{E0L}}$, then $(\tau_1, \tau_2\mathrm{T}) \in \mathrm{CUBE}_{\mathrm{ET0L}}$, $\tau_1 \in \mathrm{TYPE}_{\mathrm{E0L}}$, and $\tau_2\mathrm{T} \in \mathrm{TYPE}_{\mathrm{ET0L}} \setminus \mathrm{TYPE}_{\mathrm{E0L}}$. Thus, the simple inclusions hold by Theorem 2.38, page 26. The case $\tau_1 \in \mathrm{TYPE}_{\mathrm{0L}}$ already has been considered by Theorem 4.9, page 43. For the remaining case $\tau_1 \in \mathrm{TYPE}_{\mathrm{E0L}} \setminus \mathrm{TYPE}_{\mathrm{0L}}$, the assertion immediately follows by Theorem 5.8. ■

Theorem 5.10 For all $(\tau_1, \tau_2) \in \mathrm{CUBE}_{\mathrm{T0L}}$ and all non-empty sets $K_1, K_2 \subseteq \mathbb{N}$ with $K_1 \subseteq K_2$ it holds that

$$\mathcal{L}(K_1\mathrm{ml}\tau_1\mathrm{0L}) \subsetneqq \mathcal{L}(K_2\mathrm{mlE}\tau_2\mathrm{0L}).$$

Proof: If $(\tau_1, \tau_2) \in \mathrm{CUBE}_{\mathrm{T0L}}$, then $(\tau_1,\mathrm{E}\tau_2) \in \mathrm{CUBE}_{\mathrm{ET0L}}$, $\tau_1 \in \mathrm{TYPE}_{\mathrm{T0L}}$, and $\mathrm{E}\tau_2 \in \mathrm{TYPE}_{\mathrm{ET0L}} \setminus \mathrm{TYPE}_{\mathrm{T0L}}$. Thus, the assertion immediately follows by Theorem 5.6 and Theorem 2.38, page 26. ■

As mentioned above, it is supposed that the corresponding reverse result of Lemma 5.5, regarding mlPDT0L languages and mlE0L languages, also is true. So this conjecture as well as its implications are noted subsequently.

Conjecture 5.11 For all non-empty sets $K_1, K_2 \subseteq \mathbb{N}$ it holds that

$$\mathcal{L}(K_1\mathrm{mlPDT0L}) \not\subseteq \mathcal{L}(K_2\mathrm{mlE0L}).\ ■$$

If Conjecture 5.11 is true then also the following two statements would hold. The first represents the corresponding reverse result of Theorem 5.6, the second an additional result regarding incomparability.

Conjecture 5.12 For all $\tau_1 \in \mathrm{TYPE}_{\mathrm{ET0L}} \setminus \mathrm{TYPE}_{\mathrm{E0L}}$, $\tau_2 \in \mathrm{TYPE}_{\mathrm{E0L}}$, and all non-empty sets $K_1, K_2 \subseteq \mathbb{N}$ it holds that

$$\mathcal{L}(K_1\mathrm{ml}\tau_1\mathrm{0L}) \not\subseteq \mathcal{L}(K_2\mathrm{ml}\tau_2\mathrm{0L}).\ ■$$

Conjecture 5.13 For all $\tau_1 \in \mathrm{TYPE}_{\mathrm{E0L}} \setminus \mathrm{TYPE}_{\mathrm{0L}}$, $\tau_2 \in \mathrm{TYPE}_{\mathrm{T0L}} \setminus \mathrm{TYPE}_{\mathrm{0L}}$, and all non-empty sets $K_1, K_2 \subseteq \mathbb{N}$, each family $\mathcal{L}(K_1\mathrm{ml}\tau_1\mathrm{0L})$ is incomparable to each family $\mathcal{L}(K_2\mathrm{ml}\tau_2\mathrm{0L})$, but they are not disjoint. ■

The following result regarding non-inclusions amongst propagating and deterministic families of mlE0L languages is proved similar to [36], Theorem 4.6.

Lemma 5.14 For all non-empty sets $K_1, K_2 \subseteq \mathbb{N}$ it holds that

$$\mathcal{L}(K_1\text{mlP0L}) \not\subseteq \mathcal{L}(K_2\text{mlED0L}).$$

Proof: Let $K_1, K_2 \subseteq \mathbb{N}$ be arbitrary non-empty sets. Then the language $L = \{a\}^+\{b\}^+$ is generated by the K_1mlP0L system $G = (\Sigma, h, ab, \kappa)$ with $\Sigma = \{a, b\}$, $h(a) = \{a, a^2\}$, $h(b) = \{b, b^2\}$, and an arbitrary mapping $\kappa : \Sigma \to K_1$.

Assume that L also is generated by any K_2mlED0L system $G' = (\Sigma', h', \omega', \Delta, \kappa')$ with $\Delta = \Sigma$. At first, consider two arbitrary derivations $\omega \Longrightarrow^n w_1$ and $\omega \Longrightarrow^n w_2$ according to G' of the same length $n \in \mathbb{N}_0$. Since G' is deterministic, it follows that $\#_x w_1 = \#_x w_2$ for all $x \in \Sigma$. Consequently, $w_1 = w_2$ if $w_1, w_2 \in L = \{a\}^+\{b\}^+$. This guarantees that every derivation starting from ω, generates any fixed terminal word $w \in L$ after the same number $n_w \in \mathbb{N}_0$ of derivation steps and thus, every such derivation contains all words of L, without repetition. $(*)$

If $a \Longrightarrow^* \varepsilon$ and $b \Longrightarrow^* \varepsilon$, then from the word $ab \in L$ the empty word $\varepsilon \notin L$ can be derived (Another contradicting consequence is that $L(G')$ would be finite). Thus, either $a \Longrightarrow^* \varepsilon$ is not possible, or $b \Longrightarrow^* \varepsilon$ is not possible, or both.

If $a \Longrightarrow^* \varepsilon$, but $b \Longrightarrow^* \varepsilon$ is not possible, there exists a derivation $ab^2 \Longrightarrow^* a^i b^j \in L$ according to G' with suitable integers $i, j \in \mathbb{N}$, where the word $a^i b^j$ arises from b^2, only. Let $a^i b^j = v_1 v_2$ where $v_1 \neq \varepsilon$ arises from the left occurrence of b and $v_2 \neq \varepsilon$ arises from the right occurrence of b. Then it follows that $v_1 \in \{a\}^+\{b\}^*$ and $v_2 \in \{a\}^*\{b\}^+$. Thus, also $ab^2 \Longrightarrow^* v_2 v_1 \in \{a\}^*\{b\}^+\{a\}^+\{b\}^*$ is a derivation according to G', a contradiction. Analogous arguments disprove the case that $b \Longrightarrow^* \varepsilon$, but $a \Longrightarrow^* \varepsilon$ is not possible.

Finally, consider the words $ab^2, a^2b \in L$. By the facts proved above (see $(*)$) it follows that either $ab^2 \Longrightarrow^* a^2b$ or $a^2b \Longrightarrow^* ab^2$. Let $ab^2 = c_1c_2c_3$ and $a^2b = c'_1c'_2c'_3$ with $c_i, c'_i \in \Delta$ for $i = 1, 2, 3$. Since neither $a \Longrightarrow^* \varepsilon$ nor $b \Longrightarrow^* \varepsilon$ is possible, the first case implies that c'_i arises from c_i for all $i = 1, 2, 3$. Thus, also $c_1c_3c_2 = ab^2 \Longrightarrow^* c'_1c'_3c'_2 = aba \notin L$ is a derivation according to G'. Analogous arguments disprove the second case which completes the proof. ∎

It remains open whether this result also can be extended to $\mathcal{L}(K_1\text{mlP0L}) \not\subseteq \mathcal{L}(K_2\text{mlEDT0L})$. For this purpose, mlP0L languages have to be found which cannot be generated by deterministic extended systems even if different tables are used. At least for the case $1 \in K_2$ this is not possible since $\mathcal{L}(\text{1lEDT0L}) = \mathcal{L}(\text{mlET0L})$ is proved later in this section (see Corollary 5.20 (e), page 78).

In the corresponding reverse case of mlD0L and mlEPT0L languages such an extended result is proved by the following Lemma.

Lemma 5.15 For all non-empty sets $K_1, K_2 \subseteq \mathbb{N}$ it holds that

$$\mathcal{L}(K_1\text{mlD0L}) \not\subseteq \mathcal{L}(K_2\text{mlEPT0L}).$$

Proof: Obviously, the language $\{a, \varepsilon\} \in \mathcal{L}(K_1\text{mlD0L})$, but $\{a, \varepsilon\} \notin \mathcal{L}(K_2\text{mlEPT0L})$, for all non-empty sets $K_1, K_2 \subseteq \mathbb{N}$. ■

As a consequence of Lemma 5.14 and Lemma 5.15, the following two theorems hold regarding incomparabilities respectively strict inclusions.

Theorem 5.16 For all non-empty sets $K_1, K_2 \subseteq \mathbb{N}$, each of the families $\mathcal{L}(K_1\text{mlP0L})$, $\mathcal{L}(K_1\text{mlPT0L})$, $\mathcal{L}(K_1\text{mlEP0L})$, and $\mathcal{L}(K_1\text{mlEPT0L})$, is incomparable to each of the families $\mathcal{L}(K_2\text{mlD0L})$, $\mathcal{L}(K_2\text{mlDT0L})$, and $\mathcal{L}(K_2\text{mlED0L})$, but they are not disjoint.

Proof: The assertion immediately follows by Lemma 5.14 and Lemma 5.15 since $\mathcal{L}(K_1\text{mlP0L})$ is a subset of each of the three families, $\mathcal{L}(K_1\text{mlPT0L})$, $\mathcal{L}(K_1\text{mlEP0L})$, and $\mathcal{L}(K_1\text{mlEPT0L})$, as well as $\mathcal{L}(K_1\text{mlD0L})$ is a subset of each of the two families, $\mathcal{L}(K_1\text{mlDT0L})$ and $\mathcal{L}(K_1\text{mlED0L})$, by Theorem 2.38, page 26. All intersections are non-empty, obviously. ■

For the same reasons as mentioned above following Lemma 5.14, it remains open whether this result also can be extended by the incomparability of the family $\mathcal{L}(K_2\text{mlEDT0L})$ and the families of propagating languages of Theorem 5.16. And also for the same reasons it is clear that at least for the case $1 \in K_2$ this is not possible (see Theorem 5.21, page 79).

Theorem 5.17 For all $(\tau_1, \tau_2) \in \text{CUBE}_{\text{E0L}} \cup \{(\text{EPT,ET}), (\text{EPDT,EDT})\}$ with $\tau_1 \neq \tau_2$, and for all non-empty sets $K_1, K_2 \subseteq \mathbb{N}$ with $K_1 \subseteq K_2$ it holds that

$$\mathcal{L}(K_1\text{ml}\tau_1\text{0L}) \subsetneqq \mathcal{L}(K_2\text{ml}\tau_2\text{0L}).$$

Proof: Let $K_1, K_2 \subseteq \mathbb{N}$ with $\emptyset \neq K_1 \subseteq K_2$. At first, consider $(\tau_1, \tau_2) \in \text{CUBE}_{\text{E0L}}$ with $\tau_1 \neq \tau_2$. Then the assertion already has been proved for the case $\tau_1, \tau_2 \in \text{TYPE}_{\text{0L}}$ by Theorem 4.9, page 43, and for the case $\tau_1 \in \text{TYPE}_{\text{0L}}$ and $\tau_2 \in \text{TYPE}_{\text{E0L}} \setminus \text{TYPE}_{\text{0L}}$ by Theorem 5.10. Thus, the following six cases of (τ_1, τ_2)

remain to be proved: (EPD,EP), (ED,E), (EPD,ED), (EP,E), (EPDT,EDT), and (EPT,ET).

The simple inclusions hold by Theorem 2.38, page 26. The reverse non-inclusions follow from Lemma 5.14 for the first two cases, and from Lemma 5.15 for the last four cases. ∎

For the same reasons as mentioned above following Lemma 5.14, it remains open whether this result also can be extended by $\mathcal{L}(K_1\text{mlEDT0L}) \subsetneqq \mathcal{L}(K_2\text{mlET0L})$ or by $\mathcal{L}(K_1\text{mlEPDT0L}) \subsetneqq \mathcal{L}(K_2\text{mlEPT0L})$. And also for the same reasons it is clear that at least for the case $1 \in K_2$ respectively $K_2 = \{1\}$ this is not possible (see Theorem 5.21, page 79).

The following Theorem 5.18 is a generalization of [36], Theorem 4.10. It states that each MmlET0L system (MmlEDT0L system) can be simulated by a KmlET0L system (KmlEDT0L system) if M consists of finite sums of limits of K. Since $\mathcal{L}(\text{1lET0L}) = \mathcal{L}(\text{1lEDT0L})$, by [36], Theorem 4.7, it follows especially that each mlET0L language is generated by a 1lET0L system as well as by a 1lEDT0L system. In the sequel, let

$$FS(K) = \{\Sigma_{i=1}^{n} k_i \mid k_1, \ldots, k_n \in K,\, n \in \mathbb{N}\}$$

denote the set of finite sums of elements of K for non-empty sets $K \subseteq \mathbb{N}$.

Theorem 5.18 For all non-empty sets $K, M \subseteq \mathbb{N}$ with $M \subseteq FS(K)$ it holds that

$$\mathcal{L}(M\text{mlET0L}) \subseteq \mathcal{L}(K\text{mlET0L}) \qquad \text{and} \qquad \mathcal{L}(M\text{mlEDT0L}) \subseteq \mathcal{L}(K\text{mlEDT0L}).$$

Proof: Consider two non-empty sets $K, M \subseteq \mathbb{N}$ with $M \subseteq FS(K)$ and let L be an MmlET0L language (respectively MmlEDT0L language). Then L is generated by an MmlET0L system (respectively MmlEDT0L system) $G_M = (\Sigma_M, H_M, \omega_M, \Delta, \kappa_M)$ which satisfies the normal form properties (a)–(d) of Theorem 5.1. Let $F_M \in \Sigma_M$ denote the failure symbol and $h_T \in H_M$ the terminal table of this normal form G_M. Since $M \subseteq FS(K)$, every limit $\kappa_M(x)$ of a symbol $x \in \Sigma_M$ can be represented as the finite sum of some fixed limits $k_{1,x}, \ldots, k_{n_x,x} \in K$ with $n_x \in \mathbb{N}$. Thus, the limiting function κ_M can be written as

$$\kappa_M = \sum_{i=1}^{n} \kappa_i \qquad \text{where} \qquad \kappa_i(x) = \begin{cases} k_{i,x} & \text{if } i \leq n_x, \\ 0 & \text{otherwise,} \end{cases} \tag{5.4}$$

for $x \in \Sigma_M$, $i = 1, \ldots, n$, and $n = \max\{n_x \mid x \in \Sigma_M\} \in \mathbb{N}$. If $n_x = 1$ for every $x \in \Sigma_M$, then $\kappa_M(x) \in K$ for all $x \in \Sigma_M$ and thus, nothing has to be proved.

Otherwise, $n_x > 1$ for at least one $x \in \Sigma_M$. For this case it is proved that G_M is simulated by the KmlET0L system (respectively KmlEDT0L system) G as described below.

The idea of this construction is that for each table $h \in H_M$ because of (5.4), each application of h_{κ_M} can be simulated by the composition of $h_{\kappa_1}, \ldots, h_{\kappa_n}$ where it has to be secured that the application of h_{κ_i} does not rewrite any occurrence which arises from a previous application of h_{κ_j}, for all $i = 2, \ldots, n$ and $j < i$.

Therefore, the simulation of a derivation step $w \overset{h}{\Longrightarrow}_{G_M} v$ is divided into n layers where, for $i = 1, \ldots, n$, the i-th layer simulates the modified application of h_{κ_i} as just described. This is reached by using the shadow alphabets $\Sigma' = \{x' \mid x \in \Sigma_M\}$ and $\Sigma^{(i)} = \{x_i \mid x \in \Sigma_M\}$, for $i = 1, \ldots, n$, as well as further non-terminal control symbols as described below. Thereby, an occurrence of a shadow symbol $x_i \in \Sigma^{(i)}$, for $i = 1, \ldots, n$, represents an occurrence of the symbol $x \in \Sigma_M$ within the word w which has not been rewritten within the first $i - 1$ layers and thus, is allowed to be rewritten within the i-th layer. An occurrence of a shadow symbol $x' \in \Sigma'$ represents an occurrence of the symbol $x \in \Sigma_M$ within the word v which has been introduced by one of the n layers and thus, is not allowed to be rewritten by one of the remaining layers. See Example 5.19 for a more detailed impression regarding the working method of this simulation.

It is interesting to note that the construction of system G also can be extended to the case that $n_x = 1$ for all $x \in \Sigma_M$.

In the sequel, let $h(x) = \{x\}$ for every unspecified value $h(x)$ with $h \in H$ and $x \in \Sigma$. Moreover, let $w^{(i)}$ (resp. w') denote the word that results from a word $w \in \Sigma^*$ by rewriting each occurrence of every symbol $x \in \Sigma_M$ by the new non-terminal shadow symbol x_i (resp. x'), for $i = 1, \ldots, n$. For example, $w = xx_1x_2x'$ with $x \in \Sigma_M$ implies that $w' = x'x_1x_2x'$, $w^{(1)} = x_1x_1x_2x'$ and $w^{(2)} = x_2x_1x_2x'$.

Let $S^{(i)} = \{w^{(i)} \mid w \in S\}$, for $i = 1, \ldots, n$, and $S' = \{w' \mid w \in S\}$ for every set $S \subseteq \Sigma^*$. Then the parameters of system $G = (\Sigma, H, \omega, \Delta, \kappa)$ are defined as

$$
\begin{aligned}
\omega &= 1\omega_M, \\
\Sigma &= \Sigma_M \cup \Sigma'_M \cup \bigcup_{i=1}^{n} \Sigma_M^{(i)} \cup \{1, \ldots, n\} \cup \{\Lambda_1, \ldots, \Lambda_n\} \cup \{S_h \mid h \in H_M\} \cup \{F\}, \\
H &= \{g_1, \ldots, g_n\} \cup \{g_E, g_R, g_T\} \cup \{h_D \mid h \in H_M\}.
\end{aligned}
$$

The set $\Sigma \backslash \Sigma_M$ consists of control symbols which control the word generating process as follows. The task of the shadow alphabets Σ' and $\Sigma^{(i)}$, for $i = 1, \ldots, n$, already has been mentioned above. The symbols $i = 1, \ldots, n$, serve as the layer counter and respectively denote that the simulation of h_{κ_i} for the i-th layer is in progress. The completion of the i-th layer is marked by introducing the control symbol Λ_i and by rewriting it in the next derivation step by $i+1$ if $i < n$, respectively by 1 if $i = n$. As soon as the first layer, i.e. the simulation of h_{κ_1}, has reached an irreversible state, the symbol S_h is introduced which signals that the simulation of h_{κ_M} still is in progress while no other table $h' \in H_M$ is allowed to be simulated. The symbol S_h is not removed until the last layer, i.e. the simulation of h_{κ_n}, has been completed. The symbol F is the *failure symbol* of G which never can be removed once it has been introduced. Thus, the symbol F marks results of "forbidden" derivations.

For $i = 1, \ldots, n$, the simulation of h_{κ_i} by the i-th layer is started by rewriting each occurrence of a symbol $x \in \Sigma_M$ by its corresponding shadow symbol $x_i \in \Sigma^{(i)}$. This task is performed by the tables g_i with

$$g_1(y) = \begin{cases} \{x_1\} & \text{if } y = x \in \Sigma_M, \\ \{F\} & \text{if } y \in \Sigma'_M \cup \Sigma_M^{(n)} \cup \{\Lambda_1, \ldots, \Lambda_n\} \cup \{2, \ldots, n\} \cup \{S_h \mid h \in H_M\}, \end{cases}$$

$$g_i(y) = \begin{cases} \{x_i\} & \text{if } y = x \in \Sigma_M, \\ \{F\} & \text{if } y \in \Sigma_M^{(i-1)} \cup \{\Lambda_1, \ldots, \Lambda_n\} \cup \{1, \ldots, n\} \setminus \{i\}, \end{cases}$$

for $i = 2, \ldots, n$. For $h \in H_M$ and $i = 1, \ldots, n$, each simulation of h_{κ_i} respectively is completed by a unique application of the *derive table* h_D followed by at least one application of the *erase table* g_E. Applied to the shadow alphabet $\Sigma^{(i)}$, for $i = 1, \ldots, n$, the derive table h_D (together with the limiting function κ) is designed to behave similar to h_{κ_i} with

$$h_D(y) = \begin{cases} (h(x))' & \text{if } y = x_i \in \Sigma_M^{(i)},\ i \leq n_x, \\ \{S_h \Lambda_i\} & \text{if } y = i,\ i = 1, \ldots, n, \\ \{\varepsilon\} & \text{if } y = S_h, \\ \{F\} & \text{if } y \in \Sigma_M \cup \{\Lambda_1, \ldots, \Lambda_n\} \cup \{S_{h'} \mid h' \in H_M \setminus \{h\}\}. \end{cases}$$

and

$$\kappa(y) = \begin{cases} k_{i,x} & \text{if } y = x_i \in \Sigma_M^{(i)},\ i \leq n_x, \\ k_0 & \text{otherwise,} \end{cases}$$

for every $y \in \Sigma$, where k_0 is an arbitrary but fixed element of K (e.g. $k_0 = \min K$).

The *erase table* g_E increments the layer counter i cyclically and erases the indices of the shadow symbols $x_i \in \Sigma_M^{(i)}$, i.e.

$$g_E(y) = \begin{cases} \{x\} & \text{if } y = x_i \in \Sigma_M^{(i)},\ i = 1, \ldots, n, \\ \{i+1\} & \text{if } y = \Lambda_i,\ i = 1, \ldots, n-1, \\ \{1\} & \text{if } y = \Lambda_n. \end{cases}$$

This either enables the application of the next layer if available, or completes the simulation of h_{κ_n} if the layer counter is reset to 1. In the latter case, the primes are removed by at least one application of the *reset table* g_R with

$$g_R(y) = \begin{cases} \{x\} & \text{if } y = x' \in \Sigma'_M, \\ \{\varepsilon\} & \text{if } y \in \{S_h \mid h \in H_M\}, \\ \{F\} & \text{if } y \in \{\Lambda_1, \ldots, \Lambda_n\} \cup \{2, \ldots, n\} \cup \bigcup_{i=1}^{n} \Sigma_M^{(i)}. \end{cases}$$

Subsequently, either the simulation of another derivation step according to G_M can be started by the table g_1, or the whole simulation is finished by the *terminal table* g_T which generates a terminal word just by removing the layer counter with

$$g_T(y) = \begin{cases} \{\varepsilon\} & \text{if } y = 1, \\ \{F\} & \text{if } y \in \Sigma \setminus (\Delta \cup \{1\}). \end{cases}$$

In order to prove that $L(G_M) = L(G)$ consider the following two schemes, (5.5) and (5.6), of derivations which are designed to generate the same terminal words according to G_M respectively according to G. The first derivation scheme according to G_M is

$$\omega_M \overset{h^{(1)}}{\Longrightarrow} w_1 \overset{h^{(2)}}{\Longrightarrow} w_2 \overset{h^{(3)}}{\Longrightarrow} \ldots \overset{h^{(r)}}{\Longrightarrow} w_r = w_0 \in \Delta^*, \tag{5.5}$$

and the second derivation scheme according to G is

$$\omega = 1\omega_M \overset{D(h^{(1)})}{\Longrightarrow} 1w_1 \overset{D(h^{(2)})}{\Longrightarrow} 1w_2 \overset{D(h^{(3)})}{\Longrightarrow} \ldots \overset{D(h^{(r)})}{\Longrightarrow} 1w_r \overset{g_T}{\Longrightarrow} w_r = w_0 \in \Delta^*, \tag{5.6}$$

both with suitable tables $h^{(j)} \in H_M$ and words $w_j \in \Sigma_M^*$ for $j = 1, \ldots, r$, $r \in \mathbb{N}_0$, where $w_1 \in h_{\kappa_M}^{(1)}(\omega_M)$ and $w_j \in h_{\kappa_M}^{(j)}(w_{j-1})$ for $j = 2, \ldots, r$. In (5.6) each formula $1w \overset{D(h)}{\Longrightarrow} 1v$ with $h \in H_M$ and $w, v \in \Sigma_M^*$, denotes the derivation scheme $D(h)$

$$1w \overset{D_1(h)}{\Longrightarrow} S_h 2v_1 \overset{D_2(h)}{\Longrightarrow} S_h 3v_2 \overset{D_3(h)}{\Longrightarrow} \ldots \overset{D_{n-1}(h)}{\Longrightarrow} S_h n v_{n-1} \overset{D_n(h)}{\Longrightarrow} S_h 1 v_n \overset{g_R}{\Longrightarrow}{}^{+}\ 1v \tag{5.7}$$

with suitable words $v_i \in (\Sigma_M \cup \Sigma'_M)^*$ with $v'_i \in (h_{\kappa_{\leq i}}(w))'$ where $\kappa_{\leq i} = \sum_{j=1}^{i} \kappa_j$, for $i = 1, \ldots, n$. The purpose of the derivation scheme $D(h)$ is to simulate a derivation

step $w \stackrel{h}{\Longrightarrow} v$ according to G_M. Furthermore, in (5.7) the formula $1w \stackrel{D_1(h)}{\Longrightarrow} S_h 2v_1$ stands for the derivation scheme $D_1(h)$

$$1w \stackrel{g_1}{\Longrightarrow}{}^* 1w^{(1)} \stackrel{h_D}{\Longrightarrow} S_h \Lambda_1 v_1^{(1)} \stackrel{g_E}{\Longrightarrow}{}^+ S_h 2v_1, \tag{5.8}$$

the formula $S_h i v_{i-1} \stackrel{D_i(h)}{\Longrightarrow} S_h(i+1)v_i$ stands for the derivation scheme $D_i(h)$

$$S_h i v_{i-1} \stackrel{g_i}{\Longrightarrow}{}^* S_h i v_{i-1}^{(i)} \stackrel{h_D}{\Longrightarrow} S_h \Lambda_i v_i^{(i)} \stackrel{g_E}{\Longrightarrow}{}^+ S_h(i+1)v_i, \tag{5.9}$$

for $i = 2, \ldots, n-1$, and finally, the formula $S_h n v_{n-1} \stackrel{D_n(h)}{\Longrightarrow} S_h 1 v_n$ stands for the derivation scheme $D_n(h)$

$$S_h n v_{n-1} \stackrel{g_n}{\Longrightarrow}{}^* S_h n v_{n-1}^{(n)} \stackrel{h_D}{\Longrightarrow} S_h \Lambda_n v_n^{(n)} \stackrel{g_E}{\Longrightarrow}{}^+ S_h 1 v_n. \tag{5.10}$$

For $i = 1, \ldots, n$, the derivation scheme $D_i(h)$ represents the i-th layer of the simulation of a derivation step $w \stackrel{h}{\Longrightarrow} v$ according to G_M. Now, using these derivation schemes the following three statements can be made:

(A) For all $h \in H_M$ and $w, v \in \Sigma_M^*$ it holds that

$$1w \stackrel{D(h)}{\Longrightarrow} 1v \qquad \text{if and only if} \qquad w \stackrel{h}{\Longrightarrow}_{G_M} v.$$

(B) For each derivation $D : 1w \Longrightarrow_G^* w_0$ with $w \in \Sigma_M^*$ and $w_0 \in \Delta^*$ there exist tables $h^{(j)} \in H_M$ and words $w_j \in \Sigma_M^*$, for $j = 1, \ldots, r$ with $r \in \mathbb{N}_0$, such that

$$1w \stackrel{D(h^{(1)})}{\Longrightarrow} 1w_1 \stackrel{D(h^{(2)})}{\Longrightarrow} 1w_2 \stackrel{D(h^{(3)})}{\Longrightarrow} \ldots \stackrel{D(h^{(r)})}{\Longrightarrow} 1w_r \stackrel{g_T}{\Longrightarrow} w_r = w_0.$$

These statements are used for the proof that $L(G_M) = L(G)$ as follows. Let $w_0 \in L(G_M)$. Then a derivation of the form (5.5) exists with suitable tables $h^{(j)} \in H_M$ and words $w_j \in \Sigma_M^*$ for $j = 1, \ldots, r$, $r \in \mathbb{N}_0$. Thus, also a derivation of the form (5.6) exists by Statement (A) and consequently $w_0 \in L(G)$.

For the opposite direction, let $w_0 \in L(G)$. Then there exists a derivation $1\omega_M \Longrightarrow_G^* w_0$. From statement (B) it follows that there exists a derivation of the form (5.6). By statement (A) this implies that also a derivation of the form (5.5) exists and consequently, $w_0 \in L(G_M)$.

Thus, only statements (A) and (B) remain to be proved.

Proof of Statement (A):

Motivated by the simulation in layers as described by (5.8), (5.9), and (5.10), consider for arbitrary $w \in \Sigma_M^*$ and $h \in H_M$ the sets

$$\begin{aligned}
M_1(w,h) &= \{v_1 \in (\Sigma_M \cup \Sigma'_M)^* \mid 1w \overset{D_1(h)}{\Longrightarrow} S_h 2v_1\},\\
M_i(w,h) &= \{v_i \in (\Sigma_M \cup \Sigma'_M)^* \mid S_h i v_{i-1} \overset{D_i(h)}{\Longrightarrow} S_h(i+1)v_i \text{ with } v_{i-1} \in M_{i-1}(w,h)\},\\
&\quad \text{for } i = 2,\ldots,n-1,\\
M_n(w,h) &= \{v_n \in (\Sigma_M \cup \Sigma'_M)^* \mid S_h n v_{n-1} \overset{D_n(h)}{\Longrightarrow} S_h 1 v_n \text{ with } v_{n-1} \in M_{n-1}(w,h)\}.
\end{aligned}$$

Then the following statement holds:

(A1) For $w \in \Sigma_M^*$, $h \in H_M$, and $i = 1,\ldots,n$, $M_i(w,h)$ equals the set of all words v_i which arise from the word w by rewriting all, but at most $\kappa_{\leq i}(x)$ occurrences of each symbol $x \in \Sigma_M$ by an arbitrary word of $(h(x))'$.

Statement (A1) is proved by induction over i. In the following, let $w \in \Sigma_M^*$ and $h \in H_M$. For the case $i = 1$ consider (5.8). Since the resulting word $S_h 2v_1$ does not contain the failure symbol F, the table h_D only can be applied after the table g_1 has converted the word $w \in \Sigma_M^*$ into the word $w^{(1)} \in (\Sigma^{(1)})^*$, completely, i.e. $1w \overset{g_1}{\Longrightarrow}{}^* 1w^{(1)}$. Thus, $1w^{(1)} \overset{h_D}{\Longrightarrow} S_h \Lambda_1 v_1^{(1)}$ with $v_1 \in (\Sigma_M \cup \Sigma'_M)^*$ and $v'_1 \in (h_{\kappa_{\leq 1}}(w))'$, since $\kappa(x_1) = k_{1,x} = \kappa_{\leq 1}(x)$ for all $x \in \Sigma_M$. Subsequently, the derivation $S_h \Lambda_1 v_1^{(1)} \overset{g_E}{\Longrightarrow}{}^+ S_h 2v_1$ completes the proof of Statement (A1) for $i = 1$.

For the case $1 < i < n$ consider (5.9) and assume that Statement (A1) holds for $i-1$. Since none of the tables g_i, h_D, and g_E is able to modify any symbol $x' \in \Sigma'_M$ and because $\kappa(x_i) = k_{i,x} = \kappa_i(x)$ if $i \leq n_x$ and $h_D(x_i) = x_i$ if $i > n_x$ for all $x \in \Sigma_M$, analogous arguments as above imply that $M_i(w,h)$ is the set of all words v_i which arise from a word $v_{i-1} \in M_{i-1}(w,h)$ by rewriting all, but at most $\kappa_i(x)$ occurrences of each symbol $x \in \Sigma_M$ by an arbitrary word of $(h(x))'$. Note that $\kappa_i(x) = 0$ if $i > n_x$. Since $\kappa_{\leq i-1}(x) + \kappa_i(x) = \kappa_{\leq i}(x)$ the induction assumption for $i-1$ completes the proof of Statement (A1) for $i = 2,\ldots,n-1$. Statement (A1) for the remaining case $i = n$ can be shown with analogous arguments with respect to (5.10).

Now, Statement (A) can be proved via the following four equivalent statements, for all $h \in H_M$ and $w, v \in \Sigma_M^*$:

(a) $1w \overset{D(h)}{\Longrightarrow} 1v$,
(b) $v' \in (M_n(w,h))'$,
(c) $v' \in (h_{\kappa_M}(w))'$,
(d) $w \overset{h}{\Longrightarrow}_{G_M} v$.

In order to prove the first equivalence, (a)$\Longleftrightarrow$(b), let $h \in H_M$ and $w, v \in \Sigma_M^*$ with $1w \overset{D(h)}{\Longrightarrow} 1v$, at first. Then (5.10) and the definition of $M_n(w,h)$ imply that there exists a word $v_n \in M_n(w,h)$ with $S_h 1v_n \overset{g_R}{\Longrightarrow}^+ 1v$. Thus, by the definition of the table g_R, which mainly rewrites occurrences $x' \in \Sigma_M'$ by x, it holds that $v_n' = v' \in M_n(w,h)'$. For the opposite implication, (b)$\Longrightarrow$(a), let $v' \in (M_n(w,h))'$. Then there exists a word $v_n \in M_n(w,h)$ with $v_n' = v'$. Thus, by the definition of $M_n(w,h)$ and the definition of the table g_R there exists a derivation of the form (5.10), i.e. $1w \overset{D(h)}{\Longrightarrow} 1v$.

The second equivalence, (b) $\Longleftrightarrow$ (c), is a direct consequence of Statement (A1) since $(M_n(w,h))' = (h_{\kappa_M}(w))'$ because $\kappa_{\leq n}(x) = \kappa_M(x)$ for all $x \in \Sigma_M$. The last equivalence, (c) $\Longleftrightarrow$ (d), is trivial and completes the proof of Statement (A).

Proof of Statement (B):

Consider the following two statements:

(B1) For all words $w, w_0 \in \Delta^*$ with $1w \Longrightarrow_G^* w_0$ it holds that

$$1w \overset{g_T}{\Longrightarrow}_G w = w_0.$$

(B2) For each derivation $D : 1w \Longrightarrow_G^* w_0$ with $w \in \Sigma_M^* \setminus \Delta^*$ and $w_0 \in \Delta^*$ there exists a word $v \in \Sigma_M^*$ and a table $h \in H_M$ such that

$$1w \overset{D(h)}{\Longrightarrow} 1v \qquad \text{and} \qquad D : 1w \Longrightarrow_G^+ 1v \Longrightarrow_G^+ w_0.$$

With the help of these statements, Statement (B) can be proved by an induction over the length $d \in \mathbb{N}_0$ of the derivation $D : 1w \Longrightarrow_G^* w_0$ with $w \in \Sigma_M^*$ and $w_0 \in \Delta^*$. For $d = 1$ it follows that $D : 1w \overset{g_T}{\Longrightarrow}_G w_0$ since the table g_T only can rewrite the symbol 1 by a terminal word within one step. Consequently, $w = w_0$ which fulfills Statement (B) for $d = 1$ with $r = 0$.

Assume that Statement (B) holds for all $d \leq d_0$ with arbitrary $d_0 \in \mathbb{N}$, and let $d = d_0 + 1$. If $w \in \Delta^*$, Statement (B1) implies that $1w \overset{g_T}{\Longrightarrow}_G w = w_0$ which fulfills Statement (B) for $d = d_0 + 1$ with $r = 0$. If $w \in \Sigma_M^* \setminus \Delta^*$ otherwise, Statement (B2)

implies that $1w \overset{D(h)}{\Longrightarrow} 1v$ and $D : 1w \Longrightarrow_G^+ 1v \Longrightarrow_G^{d_1} w_0$ for a word $v \in \Sigma_M^*$, a table $h \in H_M$, and $1 \leq d_1 \leq d_0$. Thus, for $d = d_0 + 1$ Statement (B) follows from the induction assumption with

$$1w \overset{D(h)}{\Longrightarrow} 1v \overset{D(h^{(1)})}{\Longrightarrow} 1w_1 \overset{D(h^{(2)})}{\Longrightarrow} 1w_2 \overset{D(h^{(3)})}{\Longrightarrow} \ldots \overset{D(h^{(r)})}{\Longrightarrow} 1w_r \overset{g_T}{\Longrightarrow} w_r = w_0.$$

Proof of Statement (B1): At first, consider the control sub-alphabet

$$\Sigma_C = \{\Lambda_1, \ldots, \Lambda_n\} \cup \{1, \ldots, n\} \cup \{S_h \mid h \in H_M\} \cup \{F\}.$$

Then no terminal word but ε can be derived according to G from a word $Q \in \Sigma_C^*$ since

$$h(\Sigma_C) \subseteq \Sigma_C^* \qquad \text{for all } h \in H. \tag{5.11}$$

Furthermore, consider for each $x \in \Delta$ the set

$$\Sigma_x = \{x, x', x_1, \ldots, x_n, F, F_M, F'_M, F_{M1}, \ldots, F_{Mn}\}.$$

Because of the normal form properties (see Theorem 5.1) it holds that for all tables $h \in H_M$ and symbols $x \in \Delta$,

$$h_D(x_i) = \begin{cases} (h_T(x))' = x' & \text{if } h = h_T, \text{ the terminal table of } G_M, \\ F'_M & \text{otherwise,} \end{cases}$$

for $i = 1, \ldots, n$. Thus, from a word $w \in \Delta^*$ no terminal word but w can be derived according to G since

$$h(\Sigma_x) \subseteq \Sigma_x \qquad \text{for all } h \in H \text{ and } x \in \Delta. \tag{5.12}$$

Together, (5.11) and (5.12) imply that a derivation $Qw \Longrightarrow_G^* w_0$ with $Q \in \Sigma_C^*$ and $w, w_0 \in \Delta^*$, only is possible if $w = w_0$. This proves (B1) since $1w \overset{g_T}{\Longrightarrow}_G w$ already follows by the definition of the terminal table g_T.

Proof of Statement (B2): Let $D : 1w \Longrightarrow_G^* w_0$ with $w \in \Sigma_M^* \setminus \Delta^*$ and $w_0 \in \Delta^*$. In the following it is proved that, without introducing the failure symbol F and not regarding ineffective derivations (i.e. $u \Longrightarrow^* u$) or equivalent derivations (i.e. different derivations starting with the same word and generating the same result), the tables $h \in H$ only can be applied to the word $1w$ according to the derivation scheme $D(h)$ (see (5.7)) until a word $1v$ has been derived with $v \in \Sigma_M^*$, i.e.

$$1w \overset{D_1(h)}{\Longrightarrow} S_h 2v_1 \overset{D_2(h)}{\Longrightarrow} S_h 3v_2 \overset{D_3(h)}{\Longrightarrow} \ldots \overset{D_{n-1}(h)}{\Longrightarrow} S_h n v_{n-1} \overset{D_n(h)}{\Longrightarrow} S_h 1 v_n \overset{g_R}{\Longrightarrow}^+ 1v$$

with suitable words $v_i \in (\Sigma_M \cup \Sigma'_M)^*$ with $v'_i \in (h_{\kappa_{\leq i}}(w))'$ for $i = 1, \ldots, n$. This directly proves $1w \overset{D(h)}{\Longrightarrow} 1v$. Furthermore, since the word $1v$ cannot be derived from the terminal word w_0 by (5.12), this also implies $D : 1w \Longrightarrow_G^+ 1v \Longrightarrow_G^+ w_0$.

Within the derivation D the tables of H are applied in the following order, not regarding ineffective or equivalent derivations (For an example see Example 5.19 below):

1. g_1: Since the word w contains at least one non-terminal symbol of the set $\Sigma_M \setminus \Delta$, the table g_1 only table which has an effect to the word $1w$ and which does not introduce the failure symbol F. Furthermore, because g_1 converts at least one symbol $x \in \Sigma_M$ into x_1, only the derivation $1w \overset{g_1}{\Longrightarrow}^+ 1w^{(1)}$ is possible. In fact, also the erase table g_E would have an effect, but this would only undo some conversions made by g_1.

2. $h_D \in H$ with $h \in H_M$: Without introducing the failure symbol F, only the derivation step $1w^{(1)} \overset{h_D}{\Longrightarrow} S_h \Lambda_1 v_1^{(1)}$ is possible with an arbitrary table h_D and a word $v_1 \in (\Sigma_M \cup \Sigma'_M)^*$ (with $v'_1 \in (h_{\kappa_1}(w))'$ by the definition of h_D and κ). The introduced symbol Λ_1 guarantees that all tables but g_E, would introduce the failure symbol F by the next derivation step. In addition, the introduced symbol S_h prevents successful applications of other derive tables different from h_D during the following derivation steps (until w has been derived to a word $v \in h_{\kappa_M}(w)$ at the end of $D(h)$).

3. g_E: The erase table g_E leaves the symbol S_h unchanged, rewrites the symbol Λ_1 by the symbol 2, and turns back occurrences of symbols x_1 within v_1 into x, for all $x \in \Sigma_M$. Therefore, if $v_1 \neq \varepsilon$, after the first application of g_E to the word $S_h \Lambda_1 v_1^{(1)}$ again every table except g_E, would introduce the failure symbol F or would be of no effect. Consequently, g_E has to be applied until each occurrence of every symbol x_1 within v_1 has been rewritten by x. This means that only the derivation $S_h \Lambda_1 v_1^{(1)} \overset{g_E}{\Longrightarrow}^+ S_h 2 v_1$ is possible, also for the case that $v_1 = \varepsilon$.

4. g_i, h_D, g_E, for $i = 2, \ldots, n$: Analogous arguments as those given so far (see step 1.–3.), imply that not regarding ineffective or equivalent derivations, only the

derivation (5.9)

$$S_h i v_{i-1} \overset{g_i}{\Longrightarrow}{}^* S_h i v_{i-1}^{(i)} \overset{h_D}{\Longrightarrow} S_h \Lambda_i v_i^{(i)} \overset{g_E}{\Longrightarrow}{}^+ S_h (i+1) v_i$$

for $i = 2, \ldots, n-1$, and finally the derivation (5.10)

$$S_h n v_{n-1} \overset{g_n}{\Longrightarrow}{}^* S_h n v_{n-1}^{(n)} \overset{h_D}{\Longrightarrow} S_h \Lambda_n v_n^{(n)} \overset{g_E}{\Longrightarrow}{}^+ S_h 1 v_n$$

is possible according to G with $v_i \in (\Sigma_M \cup \Sigma'_M)^*$, for $i = 2, \ldots, n$.

5. g_R: If $v_n \neq \varepsilon$, only the reset table g_R has an effect on the word $S_h 1 v_n$ without introducing the failure symbol F. As long as there still are some occurrences of symbols $x' \in \Sigma'_M$ within the word v_n, only g_R can be applied successfully. This leads to the derivation $S_h 1 v_n \overset{g_R}{\Longrightarrow}{}^+ 1v$ with $v \in \Sigma_M^*$ and $v' = v'_n$.

   If $v_n \neq \varepsilon$, additionally the derivation $S_h 1 \overset{h_D}{\Longrightarrow} S_h \Lambda_1$ is possible. By the facts proved so far (see step 1.–5., first paragraph) without introducing the failure symbol F, this derivation only can be continued to the ineffective derivation $S_h 1 \overset{h_D}{\Longrightarrow} S_h \Lambda_1 \Longrightarrow^* S_h 1$. Therefore, this case can be omitted which completes the proof of statement (B2). ■

Example 5.19 Let $K = \{1, 2\}$ and $M = \{3, 5\} \subseteq FS(K)$. Consider an MmlET0L system $G_M = (\Sigma_M, H_M, \omega_M, \Delta, \kappa_M)$ of normal form with $\Delta = \{x, y\}$,

$$\begin{aligned} \kappa_M(x) = 3 = 2 + 1 \quad &= k_{1,x} + k_{2,x} \quad &(n_x = 2), \\ \kappa_M(y) = 5 = 2 + 2 + 1 &= k_{1,y} + k_{2,y} + k_{3,y} \quad &(n_y = 3 = n). \end{aligned}$$

Thus, κ_M can be written as $\kappa_M = \kappa_1 + \kappa_2 + \kappa_3$ where $\kappa_i(z) = k_{i,z}$, for $z = x, y$ and $i = 1, 2$, $\kappa_3(x) = 0$, and $\kappa_3(y) = k_{3,y}$. According to the proof of Theorem 5.18, the system G_M can be simulated by the KmlET0L system $G = (\Sigma, H, \omega, \Delta, \kappa)$ with $\kappa(z_i) = k_{i,z}$, for $z = x, y$ and $i = 1, 2$, $\kappa(y_3) = k_{3,y}$, and $\kappa(a) = \min(K) = 1$ for all remaining symbols $a \in \Sigma$. As an example, consider the derivation

$$D: \; x^4 y \overset{h}{\Longrightarrow}_{G_M} v_1 x v_2 v_3 v_4$$

for a table $h \in H_M$, words $v_1, v_2, v_3 \in h(x)$, and a word $v_4 \in h(y)$. The simulation of the derivation D works as follows:

Layer for h_{κ_1}:	$1x^4y$	$\xRightarrow{g_1}$	1	x_1	x	x	x	y_1
		$\xRightarrow{g_1}$	1	x_1	x_1	x	x	y_1
		$\xRightarrow{g_1}$	1	x_1	x_1	x_1	x	y_1
		$\xRightarrow{g_1}$	1	x_1	x_1	x_1	x_1	y_1
		$\xRightarrow{h_D}$	$S_h\Lambda_1$	v'_1	x_1	v'_2	x_1	v'_4
		$\xRightarrow{g_E}$	S_h2	v'_1	x	v'_2	x	v'_4
Layer for h_{κ_2}:		$\xRightarrow{g_2}$	S_h2	v'_1	x_2	v'_2	x	v'_4
		$\xRightarrow{g_2}$	S_h2	v'_1	x_2	v'_2	x_2	v'_4
		$\xRightarrow{h_D}$	$S_h\Lambda_2$	v'_1	x_2	v'_2	v'_3	v'_4
		$\xRightarrow{g_E}$	S_h3	v'_1	x	v'_2	v'_3	v'_4
Layer for h_{κ_n}:		$\xRightarrow{g_3}$	S_h3	v'_1	x_3	v'_2	v'_3	v'_4
		$\xRightarrow{h_D}$	$S_h\Lambda_3$	v'_1	x_3	v'_2	v'_3	v'_4
		$\xRightarrow{g_E}$	S_h1	v'_1	x	v'_2	v'_3	v'_4
Finish of h_{κ_M}:		$\xRightarrow{g_R}$	1	v'_1	x	v'_2	v'_3	v'_4
		$\xRightarrow{g_R}{}^*$	1	v_1	x	v_2	v_3	v_4
Finish of simulation:		$\xRightarrow{g_T}$		v_1	x	v_2	v_3	v_4 ■

It is not known whether Theorem 5.18 also holds for propagating systems. Since the proof of Theorem 5.18 essentially depends on the deletion of control symbols, it is not applicable to propagating systems.

Theorem 5.18 implies the following results regarding inclusion relations amongst families of mlET0L languages as well as between families of mlET0L and klET0L language families.

Corollary 5.20 Let $K, K' \subseteq \mathbb{N}$ be non-empty sets. Then

(a) $\mathcal{L}(FS(K)\text{mlET0L}) = \mathcal{L}(K\text{mlET0L})$ and
$\mathcal{L}(FS(K)\text{mlEDT0L}) = \mathcal{L}(K\text{mlEDT0L})$,

(b) $\mathcal{L}(K\text{mlET0L}) = \mathcal{L}(K'\text{mlET0L})$ and $\mathcal{L}(K\text{mlEDT0L}) = \mathcal{L}(K'\text{mlEDT0L})$ if $FS(K) = FS(K')$,

(c) $\mathcal{L}(K\text{mlET0L}) \subseteq \mathcal{L}(1\text{lEDT0L})$,

(d) $\mathcal{L}(K\text{mlET0L}) = \mathcal{L}(K\text{mlEDT0L}) = \mathcal{L}(1\text{lEDT0L})$ if $1 \in K$,

(e) $\mathcal{L}(\text{mlEDT0L}) = \mathcal{L}(\text{mlET0L}) = \mathcal{L}(\text{1lEDT0L}) = \mathcal{L}(\text{1lET0L})$
$= \mathcal{L}(\text{lEDT0L}) = \mathcal{L}(\text{lET0L})$,

(f) $\mathcal{L}((k_1 \cdot k_2)\text{lET0L}) \subseteq \mathcal{L}(k_1\text{lET0L})$ and $\mathcal{L}((k_1 \cdot k_2)\text{lEDT0L}) \subseteq \mathcal{L}(k_1\text{lEDT0L})$ for all $k_1, k_2 \in \mathbb{N}$. (Originated from [36], Theorem 4.10)

Proof: Consider arbitrary non-empty sets $K, K' \subseteq \mathbb{N}$.

(a) Theorem 5.18 implies that on the one hand,

$$\mathcal{L}(K\text{mlET0L}) \subseteq \mathcal{L}(FS(K)\text{mlET0L})$$

since $\hat{M} = K \subseteq FS(\hat{K})$ with $\hat{K} = FS(K)$, on the other hand

$$\mathcal{L}(FS(K)\text{mlET0L}) \subseteq \mathcal{L}(K\text{mlET0L})$$

since $\hat{M} = FS(K) \subseteq FS(\hat{K})$ with $\hat{K} = K$. These arguments also hold for the corresponding EDT0L languages.

(b) If $FS(K) = FS(K')$, item (a) implies that

$$\mathcal{L}(K\text{mlET0L}) = \mathcal{L}(FS(K)\text{mlET0L}) = \mathcal{L}(FS(K')\text{mlET0L}) = \mathcal{L}(K'\text{mlET0L}).$$

Analogously, this also holds for the corresponding EDT0L languages.

(c) Since $FS(1) = \mathbb{N}$, Theorem 5.18 implies that $\mathcal{L}(K\text{mlET0L}) \subseteq \mathcal{L}(\text{1lET0L})$. Because $\mathcal{L}(\text{1lET0L}) = \mathcal{L}(\text{1lEDT0L})$ by [36], Theorem 4.7, the assertion follows.

(d) If $1 \in K$, then $FS(K) = FS(1) = \mathbb{N}$ and consequently, by item (b) and [36], Theorem 4.7,

$$\mathcal{L}(K\text{mlET0L}) = \mathcal{L}(\text{1lET0L}) = \mathcal{L}(\text{1lEDT0L}) = \mathcal{L}(K\text{mlEDT0L}).$$

(e) Immediately follows by item (d) and [36], Theorem 4.7 and Theorem 4.11.

(f) Immediately follows from Theorem 5.18 since $k_1 \cdot k_2 \in FS(k_1)$, for all $k_1, k_2 \in \mathbb{N}$.

■

The following Theorem summarizes the results regarding inclusions between the families of mlET0L languages which types are of $\text{CUBE}_{\text{ET0L}} \setminus \text{CUBE}_{\text{T0L}}$.

Theorem 5.21 $\mathcal{L}(K_1\text{ml}\tau_1\text{0L}) \subsetneqq \mathcal{L}(K_2\text{ml}\tau_2\text{0L})$ for all $(\tau_1, \tau_2) \in \text{CUBE}_{\text{ET0L}} \setminus \text{CUBE}_{\text{T0L}}$ and all non-empty sets $K_1, K_2 \subseteq \mathbb{N}$ with $K_1 \subseteq K_2$, except the cases $\tau_1 = \tau_2$, $(\tau_1, \tau_2) = (\text{EDT,ET})$, and $(\tau_1, \tau_2) = (\text{EPDT,EPT})$. Furthermore,

$$\mathcal{L}(\{1\}\text{mlEPDT0L}) = \mathcal{L}(\{1\}\text{mlEPT0L}) \subsetneqq \mathcal{L}(K_1\text{mlEDT0L}) = \mathcal{L}(K_2\text{mlET0L})$$

for all non-empty sets $K_1, K_2 \subseteq \mathbb{N}$ with $1 \in K_1, K_2$.

Proof: Let $K_1, K_2 \subseteq \mathbb{N}$ with $\emptyset \neq K_1 \subseteq K_2$. At first, consider the case that $(\tau_1, \tau_2) \in \text{CUBE}_{\text{ET0L}} \setminus \text{CUBE}_{\text{T0L}}$ with $\tau_1 \neq \tau_2$ and $(\tau_1, \tau_2) \notin \{(\text{EDT,ET}), (\text{EPDT,EPT})\}$. Then this case can be partitioned into the following sub cases which have been proved in the following theorems:

$(\tau_1, \tau_2) \in \text{TYPE}_{\text{E0L}} \times (\text{TYPE}_{\text{ET0L}} \setminus \text{TYPE}_{\text{E0L}})$:	see Theorem 5.9, page 64.
$(\tau_1, \tau_2) \in \text{TYPE}_{\text{T0L}} \times (\text{TYPE}_{\text{ET0L}} \setminus \text{TYPE}_{\text{T0L}})$:	see Theorem 5.10, page 65.
$(\tau_1, \tau_2) \in \text{TYPE}_{\text{E0L}} \times \text{TYPE}_{\text{E0L}}$:	see Theorem 5.17, page 67.
$(\tau_1, \tau_2) \in \{(\text{EPT,ET}), (\text{EPDT,EDT})\}$:	see Theorem 5.17, page 67.

Furthermore, Theorem 4.7 of [36] states that

$$\mathcal{L}(\text{1lEPDT0L}) = \mathcal{L}(\text{1lEPT0L}) \subsetneqq \mathcal{L}(\text{1lEDT0L}) = \mathcal{L}(\text{1lET0L})$$

Thus, for all non-empty sets $K_1, K_2 \subseteq \mathbb{N}$ with $1 \in K_1, K_2$, Corollary 5.20 (d) implies

$$\mathcal{L}(\{1\}\text{mlEPDT0L}) = \mathcal{L}(\{1\}\text{mlEPT0L}) \subsetneqq \mathcal{L}(K_1\text{mlEDT0L}) = \mathcal{L}(K_2\text{mlET0L}). \blacksquare$$

5.2.2 Comparison with the Families of ET0L Languages

This section notes two short results regarding the inclusion relations between the families of non-limited and multi-limited ET0L systems.

Theorem 5.22 $\mathcal{L}(\text{E}\tau\text{T0L}) \subsetneqq \mathcal{L}(K\text{mlE}\tau\text{T0L})$ for all $\tau \in \text{TYPE}_{\text{0L}}$ and every non-empty set $K \subseteq \mathbb{N}$.

Proof: From [36], Theorem 4.3, it follows that $\mathcal{L}(\text{E}\tau\text{T0L}) \subsetneqq \mathcal{L}(k\text{lE}\tau\text{T0L}) \subseteq \mathcal{L}(K\text{mlE}\tau\text{T0L})$ for every non-empty set $K \subseteq \mathbb{N}$, all $k \in K$, and all $\tau \in \text{TYPE}_{\text{0L}}$. $\blacksquare$

Theorem 5.23 $\mathcal{L}(\tau_1\text{0L}) \not\subseteq \mathcal{L}(K\text{mlE}\tau_2\text{0L})$ for all $\tau_1 \in \text{TYPE}_{\text{ET0L}}$, all $\tau_2 \in \text{TYPE}_{\text{0L}}$, and every non-empty set $K \subseteq \mathbb{N}$.

Proof: Consider the language $L = \{a^{2^n} \mid n \in \mathbb{N}_0\}$. Then $L \in \mathcal{L}(\text{PD0L}) \subseteq \mathcal{L}(\tau_1\text{0L})$ for all $\tau_1 \in \text{TYPE}_{\text{ET0L}}$ by Example 2.18 (a), page 16. But $L \notin \mathcal{L}(K\text{mlE0L}) \supseteq \mathcal{L}(K\text{mlE}\tau_2\text{0L})$ for all $\tau_2 \in \text{TYPE}_{\text{0L}}$ and every non-empty set $K \subseteq \mathbb{N}$ by Example 5.4 (a), page 63 (as a consequence of the Weak Iteration Theorem, Theorem 5.3).
$\blacksquare$

5.2.3 Comparison with the Chomsky Hierarchy

It will be shown in this section that almost all the inclusion relations, which are known already between the families of klET0L languages and the Chomsky Hierarchy, also hold for KmlET0L languages.

The first three theorems regard to the families of mlE0L languages. Subsequently, the families of mlET0L languages are investigated.

Theorem 5.24 $\mathcal{L}(\text{cf}) \subsetneqq \mathcal{L}(K\text{mlE0L})$ and $\mathcal{L}(\text{cs}) \not\subseteq \mathcal{L}(K\text{mlE0L})$ for every non-empty set $K \subseteq \mathbb{N}$.

Proof: The first result follows from [32], Theorem 3.1, since

$$\mathcal{L}(\text{cf}) \subsetneqq \mathcal{L}(k\text{lE0L}) \subseteq \mathcal{L}(K\text{mlE0L})$$

for every non-empty set $K \subseteq \mathbb{N}$ and all $k \in K$. The second result holds because the context-sensitive language $\{a^{2^n} \mid n \in \mathbb{N}_0\}$ is not a member of $\mathcal{L}(K\text{mlE0L})$ for every non-empty set $K \subseteq \mathbb{N}$ (see Example 5.4, page 63). ■

By [32], Theorem 3.1, also the following result obviously is valid.

Theorem 5.25 $\mathcal{L}(\text{cf–}\varepsilon) \subsetneqq \mathcal{L}(K\text{mlEP0L})$ for every non-empty set $K \subseteq \mathbb{N}$ where $\mathcal{L}(\text{cf–}\varepsilon)$ denotes the family of all ε-free context-free languages. ■

Theorem 5.26 $\mathcal{L}(\text{reg}) \not\subseteq \mathcal{L}(K\text{mlED0L}) \subsetneqq \mathcal{L}(\text{cs})$ for every non-empty set $K \subseteq \mathbb{N}$.

Proof: The non-inclusion holds since the regular language $\{a\}^+\{b\}^+$ is not a member of $\mathcal{L}(K\text{mlED0L})$ for every non-empty set $K \subseteq \mathbb{N}$ (see the proof of Lemma 5.14, page 66).

The proof of the strict inclusion can directly be carried over from the corresponding proof of $\mathcal{L}(k\text{lED0L}) \subsetneqq \mathcal{L}(\text{cs})$ in [42]. For a given KmlED0L system $G = (\Sigma, h, \omega, \Delta, \kappa)$ with non-empty set $K \subseteq \mathbb{N}$, just the following replacements have to be carried out.

Replace each occurrence of the term "kn" by the term "$\sum_{a\in\Sigma} \kappa(a)$". Furthermore, the references "Theorem 2 of [4]", "Theorem 4.9 of [2]", "[2], p. 282", and "[3], Example 1", have to be replaced by the references to Theorem 5.29 (see below), Theorem 5.1 (Normal Form Theorem), Theorem 5.28 (see below), and Example 5.4, respectively. ■

Theorem 5.27 $\mathcal{L}(\text{cf}) \subsetneqq \mathcal{L}(K\text{mlET0L}) \not\subseteq \mathcal{L}(\text{rec})$ for every non-empty set $K \subseteq \mathbb{N}$.

Proof: The strict inclusion holds since $\mathcal{L}(\text{cf}) \subseteq \mathcal{L}(\text{ET0L}) \subsetneqq \mathcal{L}(K\text{mlET0L})$ by [36], Theorem 4.3, for every non-empty set $K \subseteq \mathbb{N}$. The non-inclusion is true since the membership for k-limited ET0L languages is not decidable, for all $k \in \mathbb{N}$ (see [8] and [9]). ■

Since $\mathcal{L}(\text{cs}) \subsetneqq \mathcal{L}(\text{rec})$ by the extended Chomsky Hierarchy (see 2.1, page 12), Theorem 5.27 immediately implies that

$$\mathcal{L}(K\text{mlET0L}) \not\subseteq \mathcal{L}(\text{cs})$$

for every non-empty set $K \subseteq \mathbb{N}$. Beyond it, the relation between $\mathcal{L}(K\text{mlET0L})$ and $\mathcal{L}(\text{cs})$ remains open.

For the propagating case, the following theorem states that each KmlEPT0L language is context-sensitive, for every non-empty set $K \subseteq \mathbb{N}$. The proof carries over the construction for the case of klET0L languages according to [36], Theorem 4.13, which bases on the construction of a type-0-grammar for the case of 1lET0L languages as given in [11].

Theorem 5.28 $\mathcal{L}(K\text{mlEPT0L}) \subseteq \mathcal{L}(\text{cs})$ for every non-empty set $K \subseteq \mathbb{N}$.

Proof: For a non-empty set $K \subseteq \mathbb{N}$ consider a KmlEPT0L system $G_\kappa = (\Sigma, H, \omega, \Delta, \kappa)$ of normal form (see Theorem 5.1, page 57). Without loss of generality, let $\Sigma \setminus \Delta = \{\omega, F, x_1, \ldots, x_m\}$ and $H = \{h_I, h_T, h_1, \ldots, h_p\}$ with $m, p \geq 1$.

Then the construction of a type-0-grammar $G = (V_N, V_T, \omega, P)$, as described for the case of klET0L languages in [36], Theorem 4.13, can be carried over as follows.

$$\begin{aligned} V_T &= \Delta, \\ V_N &= \Sigma \setminus \Delta \cup \{M_r^{(i,j)}, x'_r, B_r \mid r = 1, \ldots, m;\ i = 0, 1;\ j = 1, \ldots, \kappa(x_r)\} \\ &\quad \cup \{T, T_t \mid t = 1, \ldots, p\} \cup \{\#, D, B\}. \end{aligned}$$

The set P of productions is given by ($r = 1, \ldots, m$; $i = 0, 1$; $j = 1, \ldots, \kappa(x_r)$; $t = 1, \ldots, p$):

(1)	$\omega \to \#w\#$	for $w \in h_I(\omega)$,
(2)	$\#x \to \#DM_1^{(0,0)}x$	for $x \in \Sigma \setminus (\Delta \cup \{\omega, F\})$,
(3)	$\#x \to Tx$	for $x \in \Sigma \setminus (\Delta \cup \{\omega, F\})$,
(4)	$M_r^{(i,j)}x \to xM_r^{(i,j)}$	for $x \in V_N \setminus \{\#, x_r\}$,
(5)	$M_r^{(i,j)}x_r \to x_rM_r^{(1,j)}$,	
(6)	$M_r^{(i,j)}x_r \to x'_rM_r^{(i,j+1)}$	for $j < \kappa(x_r)$,
(7)	$M_r^{(i,j)}\# \to B_r\#$	for $i = 0$ or $j = \kappa(x_r)$,
(8)	$M_r^{(1,j)}\# \to F\#$	for $j < \kappa(x_r)$,
(9)	$xB_r \to B_rx$,	for $x \in V_N \setminus \{D\}$,
(10)	$DB_r \to DM_{r+1}^{(0,0)}$	for $r < m$,
(11)	$DB_m \to DT_t$,	
(12)	$T_tx \to xT_t$	for $x \in \Sigma \setminus (\Delta \cup \{\omega, F\})$,
(13)	$T_tx'_r \to \alpha_rT_t$	for $\alpha_r \in h_t(x_r)$,
(14)	$T_t\# \to B\#$,	
(15)	$xB \to Bx$	for $x \in V_N \setminus \{D\}$,
(16)	$DB \to \varepsilon$,	
(17)	$\#\# \to \varepsilon$	(never applicable since G_κ is propagating),
(18)	$Tx \to aT$	for $x \in \Sigma \setminus (\Delta \cup \{\omega, F\})$, $h_T(x) = \{a\}$,
(19)	$T\# \to \varepsilon$.	

In the same way as described in [36], page 282–283, whereby k has to be replaced by $\kappa(x_r)$, it follows that $L(G) = L(G_\kappa)$ and also that $|w_i| \leq |w| + 4$, $i = 0, \ldots, m$, for all derivations

$$\omega = w_0 \Longrightarrow_G w_1 \Longrightarrow_G \cdots \Longrightarrow_G w_m = w \in L(G).$$

Thus, the workspace theorem (see [29], Part III, Theorem 10.1) implies that $L(G) = L(G_\kappa)$ is context-sensitive. ■

5.2.4 Decidability Results

It is obvious that all undecidability results concerning klET0L systems (see [8], [9], and [32], for example) also are valid for the respective KmlET0L systems with $k \in K \subseteq \mathbb{N}$. In this section it is shown that all the known decidability results concerning klED0L systems as proved in [33], analogously hold for mlED0L systems.

At first, the main theorem, Theorem 2 in [33], which constitutes the basis for all the decidability results concerning klED0L systems is carried over to mlED0L systems. For the formulation of this theorem the following terms are introduced. The mapping $\psi : \Sigma^* \to \mathbb{N}_0^n$ with $\Sigma = \{a_1, \ldots, a_n\}$, $n \in \mathbb{N}$, and

$$\psi(w) = (\#_{a_1} w, \ldots, \#_{a_n} w)$$

is called the *Parikh mapping* and its values are *Parikh vectors*. The *Parikh set* of a language L over Σ is given by $\psi(L) = \{\psi(w) \mid w \in L\}$. The *alphabet of a word* w over Σ and simultaneously the alphabet of its Parikh vector $\psi(w)$ is defined as

$$\mathrm{alph}(w) = \mathrm{alph}(\psi(w)) = \{a \mid \#_a w > 0,\ a \in \Sigma\}.$$

Theorem 5.29 For a non-empty set $K \subseteq \mathbb{N}$, let $G = (\Sigma, h, \omega, \Delta, \kappa)$ be a KmlED0L system and $n = |\Sigma|$. Let

$$v_1 \Longrightarrow v_2 \Longrightarrow v_3 \Longrightarrow \ldots$$

be the uniquely determined sequence of Parikh vectors of any derivation according to G. Then there exists an algorithm which returns natural numbers $i, j \in \mathbb{N}$ and a vector $v \in \mathbb{N}_0^n$ such that

$$v_{i+r+\mu j} = v_{i+r} + \mu v \quad \text{and} \quad \mathrm{alph}(v_{i+r+\mu j}) = \mathrm{alph}(v_{i+r})$$

for all $\mu \in \mathbb{N}$ and all $r = 0, \ldots, j-1$.

Proof: The proof of [33], Theorem 2, can directly be carried over if the term "nk" (which is a typing error and actually means "$j \cdot nk$") is replaced by the term "$j \cdot \sum_{\nu=1}^{n} \kappa(a_\nu)$" and both occurrences of the term "jk" are replaced by the term "$j \cdot \kappa(a_\nu)$". ■

With the help of Theorem 5.29 the various decidability results of [33] including their proofs can be carried over to mlED0L systems directly just by replacing all the references to Theorem 2, corollaries 3 and 4, and theorems 5–7 of [33] by the references to the corresponding results of this section, i.e. Theorem 5.29, corollaries 5.30 and 5.31, and theorems 5.32–5.34, respectively.

A set $S \subseteq \mathbb{N}_0^n$ with $n \in \mathbb{N}$ is termed *linear* if there exists $r \in \mathbb{N}_0$ and $v_0, \ldots, v_r \in \mathbb{N}_0^n$ such that

$$S = \{v_0 + \sum_{i=1}^{r} \mu_i v_i \mid \mu_i \in \mathbb{N}_0,\ i = 1, \ldots, r\}.$$

A set is called *semilinear* if it is a finite union of linear sets.

Corollary 5.30 For a non-empty set $K \subseteq \mathbb{N}$, let G be a KmlED0L system. Then $\psi(L(G))$ is semilinear and effectively constructible. ■

Two languages L_1 and L_2 are *letter-equivalent* if for each $w_1 \in L_1$ there exists a $w_2 \in L_2$ with $\psi(w_1) = \psi(w_2)$, and vice versa.

Corollary 5.31 For every KmlED0L language with non-empty set $K \subseteq \mathbb{N}$ there exists a letter-equivalent regular language. ■

Theorem 5.32 For every KmlED0L system G with non-empty set $K \subseteq \mathbb{N}$ it is decidable whether $L(G)$ is finite or whether $L(G) = \emptyset$. ■

Two mlED0L systems, G with the axiom ω and G' with the axiom ω', are *growth equivalent* if all the words generated from ω or ω' according to G or G' by the same number of derivation steps, have the same length.

Theorem 5.33 Let G be a KmlED0L system and G' be a K'mlED0L system with non-empty sets $K, K' \subseteq \mathbb{N}$. Then it is decidable whether G and G' are growth equivalent. ■

Two languages $L, L' \subseteq \Sigma^*$ are *Parikh equivalent* if $\psi(L) = \psi(L')$.

Theorem 5.34 Let G be a KmlED0L system and G' be a K'mlED0L system with non-empty sets $K, K' \subseteq \mathbb{N}$. Then it is decidable whether $L(G)$ and $L(G')$ are Parikh equivalent. ■

Theorem 5.35 For every KmlED0L system $G = (\Sigma, h, \omega, \Delta, \kappa)$ with non-empty set $K \subseteq \mathbb{N}$ and every $w \in \Delta^*$ it is decidable whether $w \in L(G)$. ■

5.3 Closure Properties

In this section, several closure properties of families of mlET0L languages are shown.

Theorem 5.36 For every non-empty set $K \subseteq \mathbb{N}$, the family $\mathcal{L}(K\text{mlET0L})$ is closed with respect to substitution.

Proof: For an arbitrary non-empty set $K \subseteq \mathbb{N}$, let the language L be generated by the KmlET0L system $G_L = (\Sigma_L, H_L, \omega_L, \Delta_L, \kappa_L)$. Furthermore, consider a substitution σ on Δ_L where each language $\sigma(a)$ is generated by the KmlET0L system $G_a = (\Sigma_a, H_a, \omega_a, \Delta_a, \kappa_a)$, for $a \in \Delta_L$. Without loss of generality, let $\Delta_L \cap \Delta_a = \emptyset$ for all $a \in \Delta_L$. This always is possible since otherwise, choose a new shadow alphabet $\widehat{\Delta_L} = \{\widehat{a} \mid a \in \Delta_L\}$ with $\widehat{\Delta_L} \cap \Delta_a = \emptyset$ for all $a \in \Delta_L$, and consider the corresponding isomorphism $\widehat{\imath} : \Delta_L \to \widehat{\Delta_L}$ with $a \mapsto \widehat{\imath}(a) = \widehat{a}$. Then $\widehat{L} = \widehat{\imath}(L)$ instead of L and $\widehat{\sigma} = \widehat{\imath}^{-1} \circ \sigma$ instead of σ can be used because $\widehat{\sigma}(\widehat{a}) = \sigma(\widehat{\imath}^{-1}(\widehat{a})) = \sigma(a)$ and consequently $\widehat{\sigma}(\widehat{L}) = \sigma(L)$.

Without loss of generality, let G and G_a be of the normal form (see Theorem 5.1). Then the language $\sigma(L)$ is generated by the KmlET0L system $G = (\Sigma, H, \omega_L, \Delta, \kappa)$ which is described subsequently.

For each symbol $a \in \Delta_L$ choose $\kappa_L(a)$ copies $G_{a,i}$ of the system G_a with $G_{a,i} = (\Sigma_{a,i}, H_{a,i}, \omega_{a,i}, \Delta_a, \kappa_{a,i})$, $\Sigma_L \cap \Sigma_{a,i} = \emptyset$, and $\Sigma_{a,i} \cap \Sigma_{a,j} = \Delta_a$, for $i, j = 1, \ldots, \kappa_L(a)$ and $i \neq j$. Moreover, choose an arbitrary limit $k_0 \in K$ (e.g. $k_0 = \min K$). Then

$$\begin{aligned}
\Sigma &= \Sigma_L \cup \bigcup_{a \in \Delta_L} \bigcup_{i=1}^{\kappa_L(a)} \Sigma_{a,i} \cup \{a'_i, a''_i \mid i = 1, \ldots, \kappa_L(a),\, a \in \Delta_L\} \cup \{F\}, \\
\Delta &= \bigcup_{a \in \Delta_L} \Delta_a, \\
H &= H'_L \cup \bigcup_{a \in \Delta_L} \bigcup_{i=1}^{\kappa_L(a)} H'_{a,i} \cup \{g_s, g_c\}
\end{aligned}$$

where the set $H'_L = \{h'_L \mid h_L \in H_L\}$ contains the expansion of each table $h_L \in H_L$ on Σ which just introduces the failure symbol F outside the original alphabet Σ_L and similarly $H'_{a,i} = \{h'_{a,i} \mid h_{a,i} \in H_{a,i}\}$ with

$$h'_{a,i}(x) = \begin{cases} h_{a,i}(x) & \text{if } x \in \Sigma_{a,i} \setminus \Delta_a, \\ \{x\} & \text{if } x \in \Delta_L \cup \Delta \cup (\bigcup_{y \in \Delta_L} \bigcup_{j=1}^{\kappa_L(y)} \Sigma_{y,j} \setminus \Sigma_{a,i}), \\ \{F\} & \text{otherwise,} \end{cases}$$

for $i = 1, \ldots, \kappa_L(a)$ and $a \in \Delta_L$. The table g_s is the *separate table* with

$$g_s(x) = \begin{cases} \{a'_i a''^{k_0}_i \mid i = 1, \ldots, \kappa_L(a)\} & \text{if } x = a \in \Delta_L, \\ \{x\} & \text{if } x \in \Delta, \\ \{F\} & \text{otherwise,} \end{cases}$$

and g_c is the *check table* with

$$g_c(x) = \begin{cases} \{\varepsilon\} & \text{if } x = a_i'', \text{ for } i = 1, \ldots, \kappa_L(a),\ a \in \Delta_L, \\ \{\omega_{a,i}\} & \text{if } x = a_i', \text{ for } i = 1, \ldots, \kappa_L(a),\ a \in \Delta_L, \\ \{x\} & \text{if } x \in \Delta \cup \Delta_L, \\ \{F\} & \text{otherwise.} \end{cases}$$

Finally, the limiting function is defined as

$$\kappa(x) = \begin{cases} \kappa_L(x) & \text{if } x \in \Sigma_L, \\ \kappa_{a,i}(x) & \text{if } x \in \Sigma_{a,i} \setminus \Delta_a, \text{ for } i = 1, \ldots, \kappa_L(a),\ a \in \Delta_L, \\ k_0 & \text{otherwise.} \end{cases}$$

In principle, the system G works as follows. Starting on the initial symbol $\omega_L \in \Sigma$, no other tables but $h_L' \in H_L'$ can be applied without introducing the failure symbol F. Thus, the system G behaves as the system G_L does until a word $w_L \in L$ has been generated. Since G_L is of normal form, no other word over the non-terminal alphabet Δ_L can be derived from w_L. Therefore, the tables g_s and g_c can be used, only.

The purpose of the tables g_s and g_c is to rewrite the occurrences $a \in \Delta_L$ within w_L by one of the initial symbols $\omega_{a,i}$, $i = 1, \ldots, \kappa_L(a)$, of a copy $G_{a,i}$ of the corresponding system G_a such that at most one occurrence of each $\omega_{a,i}$ is introduced. Thereby at first, the separate table g_s tries to separate at most $\kappa_L(a)$ occurrences of the same symbol $a \in \Delta_L$ within w_L in a non-deterministic way by rewriting each occurrence uniquely by a word $a_i' a_i''^{k_0}$, $i = 1, \ldots, \kappa_L(a)$.

Then the check table g_c checks whether the non-deterministic separation of the table g_s was successful by erasing the shadow symbols a_i'' and by changing a_i' into $\omega_{a,i}$, for $i = 1, \ldots, \kappa_L(a)$. If g_s was not successful, some occurrences of a_i'' will remain and thus, each table will introduce the failure symbol F within the next step. But if g_s was successful, g_c deletes all occurrences of a_i'' within one step.

Subsequently, at most the tables $h_{a,i}' \in H_{a,i}'$, for $i = 1, \ldots, \kappa_L(a)$ and $a \in \Delta_L$, can be applied without introducing the failure symbol F. Therefore, on each initial symbol $\omega_{a,i}$ the system G now behaves as the system G_a does until a word $w_a \in \sigma(a)$ has been generated. Since each copy $G_{a,i}$ is of normal form, no other word over Δ can be derived from w_a.

As soon as all non-terminal symbols of $\Sigma_{a,i} \setminus \Delta_a$ have been derived to a terminal word, for all $i = 1, \ldots, \kappa_L(a)$ and $a \in \Delta_L$, the perhaps remaining symbols of Δ_L

only can be processed in the same way as described above starting with the separate table g_s. ■

Example 5.37 According to the proof of Theorem 5.36, consider a language L, a corresponding KmlET0L system $G_L = (\Sigma_L, H_L, \omega_L, \Delta_L, \kappa_L)$, a substitution σ on Δ_L as well as the belonging KmlET0L systems $G_{a,i} = (\Sigma_{a,i}, H_{a,i}, \omega_{a,i}, \Delta_a, \kappa_{a,i})$, for $i = 1, \ldots, \kappa_L(a)$ and $a \in \Delta_L$. Especially, let $K = \{2, 3, 4\}$, $k_0 = 2$, $w_L = a^4b^4 \in L$, $\kappa_L(a) = 3$, $\kappa_L(b) = 4$, and finally $w_a^{(j)} \in \sigma(a)$ and $w_b^{(j)} \in \sigma(b)$, for $j = 1, 2, 3, 4$.

At first, an example of a derivation according to the resulting system G with a successful non-deterministic rewriting by the table g_s is demonstrated:

$$
\begin{aligned}
\omega_L &\overset{H'_L}{\Longrightarrow}{}^{+}\ w_L = a^4b^4 \\
&\overset{g_s}{\Longrightarrow}\ a'_1a''^2_1\, a'_2a''^2_2\, a'_3a''^2_3\, a\, b'_1b''^2_1 \cdots b'_4b''^2_4 \\
&\overset{g_c}{\Longrightarrow}\ \omega_{a,1}\, \omega_{a,2}\, \omega_{a,3}\, a\, \omega_{b,1} \cdots \omega_{b,4} \\
&\overset{H'_{x,i}}{\Longrightarrow}{}^{+}\ w_a^{(1)}\, w_a^{(2)}\, w_a^{(3)}\, a\, w_b^{(1)} \cdots w_b^{(4)} \qquad \text{where } x = a, b \text{ and } i = 1, \ldots, \kappa_L(x) \\
&\overset{g_s}{\Longrightarrow}\ w_a^{(1)}\, w_a^{(2)}\, w_a^{(3)} a'_2a''^2_2 w_b^{(1)} \cdots w_b^{(4)} \\
&\overset{g_c}{\Longrightarrow}\ w_a^{(1)}\, w_a^{(2)}\, w_a^{(3)} \omega_{a,2} w_b^{(1)} \cdots w_b^{(4)} \\
&\overset{H'_{a,2}}{\Longrightarrow}{}^{+}\ w_a^{(1)}\, w_a^{(2)}\, w_a^{(3)}\, w_a^{(4)} w_b^{(1)} \cdots w_b^{(4)} \qquad \in \sigma(a^4b^4)
\end{aligned}
$$

The following example presents a derivation according to G which cannot generate a terminal word because of an unsuccessful non-deterministic rewriting by the table g_s:

$$
\begin{aligned}
\omega_L \overset{H'_L}{\Longrightarrow}{}^{+}\ w_L = a^4b^4 &\overset{g_s}{\Longrightarrow} a'_1a''^2_1\, a'_1a''^2_1\, a'_3a''^2_3\, a\, b'_1b''^2_1 \cdots b'_4b''^2_4 \\
&\overset{g_c}{\Longrightarrow} \omega_{a,1}\, \underline{\omega_{a,1}a''^2_1}\, \omega_{a,3}\, a\, \omega_{b,1} \cdots \omega_{b,4} \qquad \notin \Delta^*
\end{aligned}
$$

Since the word generated at last contains both underlined symbols, $\omega_{a,1}$ and a''^2_1, any table of the system G would introduce the failure symbol F in the next derivation step. ■

Using the previous Theorem 5.36, the following Theorem 5.38 easily proves further closure properties of KmlET0L languages (see [36], Theorem 4.14). For the idea of a direct proof of Theorem 5.38 without using the closure property with respect to substitution see [36], Theorem 4.14, as well as [26], Theorem 1.7.

Theorem 5.38 For every non-empty set $K \subseteq \mathbb{N}$, the family $\mathcal{L}(K\text{mlET0L})$ is closed with respect to (a) union, (b) concatenation, (c) homomorphism, (d) ε-free iteration, (e) iteration, and (f) mirror image.

Proof: Theorem 5.36 trivially implies the item (c) and furthermore, directly proves the items (a), (b), (d), and (e), since the regular languages $\{a, b\}$, $\{ab\}$, $\{a\}^+$, and $\{a, b\}^*$ are KmlET0L languages, respectively, for every non-empty set $K \subseteq \mathbb{N}$. The proof of the item (f) is analogous to Theorem 4.26, page 55. ■

It is interesting to note that the closure property of $\mathcal{L}(K\text{mlET0L})$ is open with respect to intersection with regular languages. If this closure property would be given, $\mathcal{L}(K\text{mlET0L})$ would be a full AFL by [29], Theorem 1.6, since $\mathcal{L}(\text{reg}) \subseteq \mathcal{L}(K\text{mlET0L})$ by Theorem 5.27 and because $\mathcal{L}(K\text{mlET0L})$ is closed with respect to substitution by Theorem 5.36, for each non-empty set $K \subseteq \mathbb{N}$.

Chapter 6

Asymptotic Inclusion Relations

In this chapter, the asymptotic behavior of the generated language of multi-limited L systems is investigated for the case that the system approximates the underlying non-limited L system by rising each value of its limiting function κ to infinity. For this purpose, for a given non-limited L system G with the alphabet Σ, an infinite sequence of respective multi-limited L systems $G_n = (G, \kappa_n)$, for $n \in \mathbb{N}$, as well as their generated languages $L(G_n)$ are considered for the case that the smallest limit $\min\{\kappa_n(a) \mid a \in \Sigma\}$ tends to infinity. The necessary basic definitions and facts of asymptotic analysis are given in Definition 6.1 and Remark 6.2 of the first section.

6.1 General Properties of mlET0L Systems

The basic definitions and facts of asymptotic analysis are given in Definition 6.1 and Remark 6.2 (refer to [7]).

Definition 6.1 Let $(X_n)_{n\in\mathbb{N}}$ be a sequence of subsets of a given set X.

(a) The *limit inferior of* $(X_n)_{n\in\mathbb{N}}$ is defined as the set of all elements which are in X_n for all except finitely many n, denoted by

$$\liminf_{n\to\infty} X_n = \bigcup_{m\geq 1} \bigcap_{n\geq m} X_n. \tag{6.1}$$

(b) The *limit superior of* $(X_n)_{n\in\mathbb{N}}$ is defined as the set of all elements which are in X_n for infinitely many n, denoted by

$$\limsup_{n\to\infty} X_n = \bigcap_{m\geq 1} \bigcup_{n\geq m} X_n. \tag{6.2}$$

(c) The sequence $(X_n)_{n\in\mathbb{N}}$ is said to *converge to the limit* $\lim_{n\to\infty} X_n$ if and only if

$$\liminf_{n\to\infty} X_n = \limsup_{n\to\infty} X_n = \lim_{n\to\infty} X_n.$$

(d) For every strictly increasing function $N : \mathbb{N} \to \mathbb{N}$, the sequence $(X_{N(n)})_{n\in\mathbb{N}}$ is called a *sub-sequence of* $(X_n)_{n\in\mathbb{N}}$. ■

Remark 6.2 Let $(X_n)_{n\in\mathbb{N}}$ be a sequence of subsets of a given set X. Then the following holds:

(a) $\liminf_{n\to\infty} X_n \subseteq \limsup_{n\to\infty} X_n$.

(b) There always exists a converging sub-sequence $(X_{N(n)})_{n\in\mathbb{N}}$ with

$$\liminf_{n\to\infty} X_{N(n)} = \limsup_{n\to\infty} X_{N(n)} = \lim_{n\to\infty} X_{N(n)}.$$

(c) If $\lim_{n\to\infty} X_n$ exists, then $\lim_{n\to\infty} X_{N(n)} = \lim_{n\to\infty} X_n$ for all sub-sequences $(X_{N(n)})_{n\in\mathbb{N}}$.

(d) For all sub-sequences $(X_{N(n)})_{n\in\mathbb{N}}$ it holds that

$$\liminf_{n\to\infty} X_n \subseteq \liminf_{n\to\infty} X_{N(n)} \qquad \text{and} \qquad \limsup_{n\to\infty} X_n \supseteq \limsup_{n\to\infty} X_{N(n)}.$$

(e) For each two sub-sequences $(X_{N(n)})_{n\in\mathbb{N}}$ and $(X_{N'(n)})_{n\in\mathbb{N}}$ it holds that

$$\liminf_{n\to\infty} X_n \subseteq \liminf_{n\to\infty} X_{N(n)} \cap \liminf_{n\to\infty} X_{N'(n)}$$

as well as

$$\limsup_{n\to\infty} X_n \supseteq \limsup_{n\to\infty} X_{N(n)} \cup \limsup_{n\to\infty} X_{N'(n)}.$$ ■

As a trivial consequence, the following corollary can be stated.

Corollary 6.3 Given an ET0L system G and, for $n \in \mathbb{N}$, arbitrary respective $K_n$mlET0L systems $G_n = (G, \kappa_n)$ with $\lim_{n\to\infty} \min K_n = \infty$. Then by the choice of K_n it always can be guaranteed that $\lim_{n\to\infty} L(G_n)$ exists, i.e.

$$\liminf_{n\to\infty} L(G_n) = \limsup_{n\to\infty} L(G_n).$$

Proof: According to Remark 6.2 (b) there always exists a converging sub-sequence $(L(G_{N(n)}))_{n\in\mathbb{N}}$. Choosing $K'_n = K_{N(n)}$ completes the proof. ■

The following theorem strengthens a result of D. Wätjen regarding k-limited T0L systems (see [36], Theorem 3.1) and extends it to multi-limited ET0L systems. It

states that each word w generated by an ET0L system G, also is generated by each respective multi-limited ET0L system (G, κ) if the smallest limit assigned by κ exceeds a certain finite bound depending on the word w.

Theorem 6.4 Given an ET0L system G and, for $n \in \mathbb{N}$, arbitrary respective $K_n$mlET0L systems $G_n = (G, \kappa_n)$ with $\lim_{n\to\infty} \min K_n = \infty$. Then

$$L(G) \subseteq \liminf_{n\to\infty} L(G_n). \tag{6.3}$$

Proof: Let $w \in L(G)$. Then there exists a derivation D_w of length $m \in \mathbb{N}_0$ with

$$D_w : w_0 = \omega \Longrightarrow_G w_1 \Longrightarrow_G \ldots \Longrightarrow_G w_m = w.$$

Let $k_w = \max\{|w_i| \mid i = 0, \ldots, m\}$, i.e. the maximal length of words of D_w. Since k_w is finite, but $\lim_{n\to\infty} \min K_n = \infty$, there exists an index $n_0 \in \mathbb{N}$ such that $\min K_n \geq k_w$ for all $n \geq n_0$. Therefore, D_w also is a derivation according to G_n, for all $n \geq n_0$, and can be written as

$$D_w : w_0 = \omega \Longrightarrow_{G_n} w_1 \Longrightarrow_{G_n} \ldots \Longrightarrow_{G_n} w_m = w.$$

Consequently, $w \in L(G_n)$ for all $n \geq n_0$, and hence,

$$w \in \bigcup_{n_0 \geq 1} \bigcap_{n \geq n_0} L(G_n) = \liminf_{n\to\infty} L(G_n). \blacksquare$$

6.2 Converging sequences of mlEPT0L and mlED0L languages

For mlEPT0L systems as well as for mlED0L systems, the following theorem states that their generative power exactly tends to the generative power of the underlying ET0L system.

Theorem 6.5 Given an ET0L system $G = (\Sigma, H, \omega, \Delta)$ and, for $n \in \mathbb{N}$, respective $K_n$mlET0L systems $G_n = (G, \kappa_n)$ with $\lim_{n\to\infty} \min K_n = \infty$. Then

$$\lim_{n\to\infty} L(G_n) = L(G) \quad \text{if} \tag{6.4}$$

(a) G is propagating, i.e. G is an (E)P(T)0L system, or

(b) G is deterministic with only one table, i.e. G is an (E)D0L system.

Proof: Because of Remark 6.2 (a) together with Theorem 6.4, for the proof of (6.4) it suffices to show that $\limsup_{n\to\infty} L(G_n) \subseteq L(G)$.

(a) Consider an arbitrary (E)P(T)0L system G and a word $w \in \limsup_{n\to\infty} L(G_n)$. Then on the one hand, for each $n \in \mathbb{N}$ there exists an index $n_0 \geq n$ such that $w \in L(G_{n_0})$. On the other hand, since $\lim_{n\to\infty} \min K_n = \infty$ there exists an index n_1 such that $\min K_n \geq |w|$ for all $n \geq n_1$. Choosing $n_0 \geq n_1$ implies that there exists a derivation D_w according to G_{n_0} such that

$$D_w : w_0 = \omega \Longrightarrow_{G_{n_0}} w_1 \Longrightarrow_{G_{n_0}} \ldots \Longrightarrow_{G_{n_0}} w_m = w$$

with $m \in \mathbb{N}_0$ and $\min K_{n_0} \geq |w|$. Since the system G and thus also the system G_{n_0} is propagating it follows that

$$\min K_{n_0} \geq |w_m| \geq |w_{m-1}| \geq \ldots \geq |w_0|.$$

Therefore, D_w also is a derivation according to G and can be written as

$$D_w : w_0 = \omega \Longrightarrow_G w_1 \Longrightarrow_G \ldots \Longrightarrow_G w_m = w.$$

Consequently, $w \in L(G)$.

(b) Consider an arbitrary (E)D0L system G and a word $w \in \limsup_{n\to\infty} L(G_n)$. Furthermore, let

$$r = \max\{|v| \mid h(a) = \{v\} \wedge a \in \Sigma\},$$
$$s = \max\{|v| \mid \omega \Longrightarrow_G^i v \wedge i = 0, \ldots, |\Sigma| - 1\}.$$

Then on the one hand, for each $n \in \mathbb{N}$ there exists an index $n_0 \geq n$ such that $w \in L(G_{n_0})$. On the other hand, since $\lim_{n\to\infty} \min K_n = \infty$ while r, s, Σ, and $|w|$ are finite, there exists an index n_1 such that $\min K_n \geq \max\{r^{|\Sigma|}, s \cdot |w|\}$ for all $n \geq n_1$. Choosing $n_0 \geq n_1$ implies that there exists a derivation D_w according to G_{n_0} such that

$$D_w : w_0 = \omega \Longrightarrow_{G_{n_0}} w_1 \Longrightarrow_{G_{n_0}} \ldots \Longrightarrow_{G_{n_0}} w_m = w$$

with $m \in \mathbb{N}_0$ and $\min K_{n_0} \geq \max\{s, r^{|\Sigma|} \cdot |w|\}$. Thus, it follows by Theorem 6.6 (d) (see below) that

$$\min K_{n_0} \geq \max\{s, r^{|\Sigma|} \cdot |w|\} \geq |w_i| \qquad \text{for all } i = 0, \ldots, m.$$

Therefore, D_w also is a derivation according to G and can be written as

$$D_w : w_0 = \omega \Longrightarrow_G w_1 \Longrightarrow_G \ldots \Longrightarrow_G w_m = w.$$

Consequently, $w \in L(G)$. ■

The following Theorem 6.6 (d) provides the technical result which is used in the proof of Theorem 6.5 (b). Items (a)–(c) of Theorem 6.6 are used for the proof of item (d), only. Nevertheless, they are listed since they may be of interest by themselves. Item (e) is a corollary of item (d). It states that in mlED0L systems, the growing of word lengths is bounded by a constant factor which depends on the respective underlying ED0L system only. This statement is of interest since it is similar to a result of Salomaa concerning 0L systems (see [29], Part VII, Theorem 13.6). The respective result concerning non-limited ED0L systems is proved in Section 6.2.1, Theorem 6.7.

Theorem 6.6 Given an ED0L system $G = (\Sigma, h, \omega, \Delta)$ and a word $w_0 \in \Sigma^*$. Let

$$r = \max\{|v| \mid h(a) = \{v\} \wedge a \in \Sigma\},$$
$$s = \max\{|v| \mid w_0 \Longrightarrow_G^i v \wedge i = 0, \ldots, |\Sigma| - 1\}.$$

Consider a respective KmlED0L system $G' = (G, \kappa)$ with $\min K \geq s$, and a sequence $(w_n)_{n\in\mathbb{N}_0}$ of words generated by a sequence $(D_n)_{n\in\mathbb{N}_0}$ of derivations according to the system G' with $D_n : w_0 \Longrightarrow_{G'} w_1 \Longrightarrow_{G'} \cdots \Longrightarrow_{G'} w_n$ for all $n \in \mathbb{N}_0$. Then the following holds:

(a) Within w_0 each occurrence of a symbol which is productive with respect to $w_{|\Sigma|}$ according to $D_{|\Sigma|}$ also is productive with respect to w_i according to D_i for all $i \geq |\Sigma|$.
(b) If there exists an index $i \geq |\Sigma|$ with $w_i \neq \varepsilon$, then $w_n \neq \varepsilon$ for all $n \in \mathbb{N}_0$.
(c) $|w_i| \leq r^i \cdot |w_j|$ for all $i, j \geq |\Sigma|$.
(d) $|w_i| \leq \max\{s, r^{|\Sigma|} \cdot |w_{n_0}|\}$ for all $i \leq n_0$ and $n_0 \in \mathbb{N}_0$ with $w_0 = \omega$ and $\max\{s, r^{|\Sigma|} \cdot |w_{n_0}|\} \leq \min K$.
(e) There exists a constant $c(G) \leq \max\{s, r^{|\Sigma|}\}$ such that

$$|w_i| \leq c(G) \cdot |w_{n_0}| \qquad \text{for all } i \leq n_0,\ n_0 \in \mathbb{N}_0$$

with $w_0 = \omega$, $w_{n_0} \neq \varepsilon$, and $\max\{s, r^{|\Sigma|} \cdot |w_{n_0}|\} \leq \min K$.

Proof:

(a) Let b_0 be an arbitrary occurrence of a symbol of Σ within w_0 which is productive with respect to $w_{|\Sigma|}$ according to $D_{|\Sigma|}$. Since $\min K \geq s$, the derivation $D_{|\Sigma|}$ also can be written as a derivation according to the non-limited system G, i.e.

$$D_{|\Sigma|} : w_0 \Longrightarrow_G w_1 \Longrightarrow_G \ldots \Longrightarrow_G w_{|\Sigma|}.$$

Therefore, there exists a derivation path (see Definition 2.39, page 27)

$$P_{|\Sigma|} : b_0 \rightarrow_G b_1 \rightarrow_G \ldots \rightarrow_G b_{|\Sigma|}$$

of length $|\Sigma|$ according to $D_{|\Sigma|}$. Because of the pigeonhole principle there exist at least two indices n_1 and n_2 with $0 \leq n_1 < n_2 \leq |\Sigma|$ and $b_{n_1} = b_{n_2}$. Thus, the derivation path $P_{|\Sigma|}$ contains a circle

$$C : c_0 = b_{n_1} \rightarrow_G c_1 \rightarrow_G \ldots \rightarrow_G c_{m-1} \rightarrow_G c_0 = b_{n_2}$$

of length $m = n_2 - n_1 > 0$ where $c_j = b_{n_1+j}$ for $j = 0, \ldots, m-1$. Since the system G is deterministic and contains only one table, each rewriting of a symbol c_j produces at least one occurrence of the symbol $c_{(j+1) \bmod m}$, for $j = 0, \ldots, m-1$. Thus, especially the occurrence b_0 within w_0 cannot be derived to ε according to G' (as well as to G) and consequently, b_0 also is productive with respect to w_i according to D_i for all $i \geq |\Sigma|$.

(b) Let $w_i \neq \varepsilon$ for an index $i \geq |\Sigma|$. Then there exists an occurrence b_0 within w_0 which is productive with respect to w_i according to D_i. By Corollary 2.41 (a), page 28, occurrence b_0 also is productive with respect to w_n according to D_i (and thus also according to D_n) for all $n \leq i$, and especially for $n = |\Sigma|$. Therefore, by item (a), the occurrence b_0 also is productive with respect to w_n according to D_n for all $n \geq |\Sigma|$. Consequently, $w_n \neq \varepsilon$ for all $n \in \mathbb{N}_0$.

(c) Let p_i, for each index $i \geq |\Sigma|$, be the set of all occurrences within w_0 which are productive with respect to w_i according to D_i. Then, by item (a) together with Corollary 2.41 (a), page 28, it follows that $p_{|\Sigma|} = p_i$ for all $i \geq |\Sigma|$. Furthermore, on the one hand, each occurrence within w_i arises from an occurrence from $p_{|\Sigma|}$, and on the other hand, each occurrence from $p_{|\Sigma|}$ contributes at most r^i occurrences to w_i, for all $i \geq |\Sigma|$. Together, this implies

$$|p_{|\Sigma|}| \leq |w_i| \leq r^i \cdot |p_{|\Sigma|}|$$

for all $i \geq |\Sigma|$. Consequently, it also holds that

$$|w_i| \leq r^i \cdot |w_j| \qquad \text{for all } i, j \geq |\Sigma|.$$

(d) Let $w_0 = \omega$ and $\min K \geq \max\{s, r^{|\Sigma|} \cdot |w_{n_0}|\}$ for an index $n_0 \in \mathbb{N}_0$. Let $M = \max\{s, r^{|\Sigma|} \cdot |w_{n_0}|\}$. Then, according to the definition of s,

$$|w_i| \leq s \leq M \qquad \text{for all } i < |\Sigma|. \tag{6.5}$$

Therefore, only the case $i \geq |\Sigma|$ and thus, $n_0 \geq |\Sigma|$ remains to be proved. This is done by induction over $i \leq n_0$ as follows. The beginning of the induction for $i < |\Sigma|$ holds by (6.5). Let $i \geq |\Sigma|$ and consider the words $w_{i-|\Sigma|}, \ldots, w_{i-1}$ which are assumed to fulfill

$$|w_j| \leq M \qquad \text{for all } j = i - |\Sigma|, \ldots, i-1.$$

Then $|w_i| \leq M$ can be shown as follows: Consider the subsequence $(w'_n)_{n \in \mathbb{N}_0}$ with $w'_n = w_{n+i-|\Sigma|}$ and let

$$s' = \max\{|v| \mid w'_0 \Longrightarrow_G^j v \wedge j = 0, \ldots, |\Sigma| - 1\}.$$

Since G is deterministic and contains only one table it follows by the induction assumption that

$$\begin{aligned} s' &= \max\{|w'_j| \mid j = 0, \ldots, |\Sigma| - 1\} \\ &= \max\{|w_j| \mid j = i - |\Sigma|, \ldots, i-1\} \\ &\leq M \leq \min K. \end{aligned}$$

Thus, by item (c) and the definition of M, it follows that

$$|w_i| = |w'_{|\Sigma|}| \leq r^{|\Sigma|} \cdot |w'_{n_0-i+|\Sigma|}| = r^{|\Sigma|} \cdot |w_{n_0}| \leq M \qquad \text{for all } i \geq |\Sigma|.$$

(e) The statement (e) immediately follows from (d) with $c(G) = \max\{s, r^{|\Sigma|}\}$ since

$$|w_i| \leq \max\{s, r^{|\Sigma|} \cdot |w_{n_0}|\} \leq \max\{s, r^{|\Sigma|}\} \cdot |w_{n_0}| \qquad \text{for all } i \leq n_0$$

if $w_{n_0} \neq \varepsilon$. ■

6.2.1 A further result concerning ED0L systems

As a further result of the previous section, the above mentioned theorem of Salomaa concerning 0L systems (see [29], Part VII, Theorem 13.6) can be expanded to ED0L systems, in a way which is similar to the proof of Theorem 6.6. Thus, also for ED0L systems the growing of word lengths is bounded by a constant factor which depends on the system parameters only.

Theorem 6.7 Given an ED0L system $G = (\Sigma, h, \omega, \Delta)$ and a word $w_0 \in \Sigma^*$. Let

$$r = \max\{|v| \mid h(a) = \{v\} \wedge a \in \Sigma\},$$
$$s = \max\{|v| \mid w_0 \Longrightarrow_G^i v \wedge i = 0, \ldots, |\Sigma| - 1\}.$$

Consider a sequence $(w_n)_{n\in\mathbb{N}_0}$ of words generated by a sequence $(D_n)_{n\in\mathbb{N}_0}$ of derivations according to system G with $D_n : w_0 \Longrightarrow_G w_1 \Longrightarrow_G \cdots \Longrightarrow_G w_n$ for all $n \in \mathbb{N}_0$. Then the following holds:

(a) Within w_0 each occurrence of a symbol which is productive with respect to $w_{|\Sigma|}$ according to $D_{|\Sigma|}$ also is productive with respect to w_i according to D_i for all $i \geq |\Sigma|$.

(b) If there exists an index $i \geq |\Sigma|$ with $w_i \neq \varepsilon$, then $w_n \neq \varepsilon$ for all $n \in \mathbb{N}_0$.

(c) $|w_i| \leq r^i \cdot |w_j|$ for all $i, j \geq |\Sigma|$.

(d) $|w_i| \leq \max\{s, r^{|\Sigma|} \cdot |w_{n_0}|\}$ for all $i \leq n_0$ and $n_0 \in \mathbb{N}_0$, if $w_0 = \omega$.

(e) There exists a constant $c(G) \leq \max\{s, r^{|\Sigma|}\}$ such that

$$|w_i| \leq c(G) \cdot |w_{n_0}| \qquad \text{for all } i \leq n_0,\ n_0 \in \mathbb{N}_0$$

with $w_0 = \omega$ and $w_{n_0} \neq \varepsilon$.

Proof:

(a) Let b_0 be an arbitrary occurrence of a symbol of Σ within w_0 which is productive with respect to $w_{|\Sigma|}$ according to $D_{|\Sigma|}$. Then there exists a derivation path

$$P_{|\Sigma|} : b_0 \rightarrow_G b_1 \rightarrow_G \ldots \rightarrow_G b_{|\Sigma|}$$

according to derivation $D_{|\Sigma|}$. Now, the proof of item (a) is completed analogously to the proof of Theorem 6.6 (a) where G' has to be replaced by G.

(b) See the proof of Theorem 6.6 (b).

(c) See the proof of Theorem 6.6 (c).

(d) Let $w_0 = \omega$ and $M = \max\{s, r^{|\Sigma|} \cdot |w_{n_0}|\}$ for an index $n_0 \in \mathbb{N}_0$. Then the definition of s implies that

$$|w_i| \leq s \leq M \qquad \text{for all } i < |\Sigma|.$$

Consequently, only the case $i \geq |\Sigma|$ and thus, $n_0 \geq |\Sigma|$ remains to be proved. Consider the subsequence $(w'_n)_{n\in\mathbb{N}_0}$ with $w'_n = w_{n+i-|\Sigma|}$. Then, by item (c) and the definition of M, it follows that

$$|w_i| = |w'_{|\Sigma|}| \leq r^{|\Sigma|} \cdot |w'_{n_0-i+|\Sigma|}| = r^{|\Sigma|} \cdot |w_{n_0}| \leq M \qquad \text{for all } i \geq |\Sigma|.$$

(e) Statement (e) immediately follows from (d) with $c(G) = \max\{s, r^{|\Sigma|}\}$ since

$$|w_i| \leq \max\{s, r^{|\Sigma|} \cdot |w_{n_0}|\} \leq \max\{s, r^{|\Sigma|}\} \cdot |w_{n_0}| \qquad \text{for all } i \leq n_0$$

if $w_{n_0} \neq \varepsilon$. ■

6.3 Non-converging sequences of mlEDT0L and mlE0L languages

Theorem 6.5 of the previous Section 6.2 has shown that each sequence $(L(G_n))_{n\in\mathbb{N}}$ of languages generated by a sequence $(G_n)_{n\in\mathbb{N}}$ of mlEPT0L systems respectively mlED0L systems with a common underlying non-limited system G, always converges to $L(G)$ if the smallest permitted limited of G_n tends to infinity.

In contrast to this result, the following Theorem 6.8 shows that such a general result does not hold for the remaining non-propagating mlEDT0L systems with at least two tables respectively for the non-propagating non-deterministic mlE0L systems. More precisely, examples of sequences $(G_n)_{n\in\mathbb{N}}$ will be given reaching each of the three possible results depending on the choice of the limiting function κ_n: $\lim_{n\to\infty} L(G_n) = L(G)$, $\lim_{n\to\infty} L(G_n) \supsetneq L(G)$, or $\lim_{n\to\infty} L(G_n)$ does not exist.

Theorem 6.8 There exists

(a) a non-propagating EDT0L system G with at least two tables as well as
(b) a non-propagating and non-deterministic E0L system G

such that for each of the following properties (i)–(iii) there exists a sequence $(G_n)_{n\in\mathbb{N}}$ of respective $K_n$mlET0L systems $G_n = (G, \kappa_n)$ with $\lim_{n\to\infty} \min K_n = \infty$ which fulfills

(i) $\lim_{n\to\infty} L(G_n) = L(G)$,
(ii) $\lim_{n\to\infty} L(G_n) \supsetneq L(G)$,
(iii) $\lim_{n\to\infty} L(G_n)$ does not exist.

Proof: The proof of the case (a) is given by the following Example 6.9. It represents a non-propagating DT0L system G as well as a sequence $(G_n)_{n\in\mathbb{N}}$ of respective $K_n$mlDT0L systems $G_n = (G, \kappa_n)$ with $\lim_{n\to\infty} \min K_n = \infty$ which fulfills the

property (i), (ii), or (iii), respectively (see item (a), (b), or (c) of Example 6.9, respectively).

The proof of the case (b) is given by the following Example 6.10. It represents a non-propagating and non-deterministic 0L system G as well as a sequence $(G_n)_{n\in\mathbb{N}}$ of respective $K_n$ml0L systems $G_n = (G, \kappa_n)$ with $\lim_{n\to\infty} \min K_n = \infty$ which fulfills the property (i), (ii), or (iii), respectively (see item (a), (b), or (c) of Example 6.10, respectively). ■

Example 6.9 Consider the non-propagating DT0L system $G = (\{a\}, \{h_1, h_2\}, a^2)$ with $h_1(a) = a^2$ and $h_2(a) = \varepsilon$, generating the language $L(G) = \{\varepsilon, a^{2^i} \mid i \in \mathbb{N}\}$, obviously. Furthermore, consider for $n \in \mathbb{N}$ respective $K_n$mlDT0L systems $G_n = (G, \kappa_n)$ with $\lim_{n\to\infty} \min K_n = \infty$. Then the following holds:

(a) $\lim_{n\to\infty} L(G_n) = L(G)$ if $K_n = \{2^n\}$.
(b) $\lim_{n\to\infty} L(G_n) = L(G) \cup \{a\}$ if $K_n = \{2^n - 1\}$.
(c) $\lim_{n\to\infty} L(G_n)$ does not exist if $K_n = \begin{cases} \{2^n\} & \text{, if } n \text{ is odd,} \\ \{2^n - 1\}, & \text{if } n \text{ is even.} \end{cases}$

Proof:

(a) If $K_n = \{2^n\}$ and $n \in \mathbb{N}$, then obviously

$$L(G_n) = \{\varepsilon, a^{2^i}, a^{2^n + m\cdot 2^n} \mid i = 1, \ldots, n \wedge m \in \mathbb{N}\}.$$

Because of Remark 6.2 (a) together with Theorem 6.4 it suffices to show that $\limsup_{n\to\infty} L(G_n) \subseteq L(G)$. Consider an arbitrary word $a^{j_0} \in \{a\}^* \setminus L(G)$ which means that $j_0 > 0$ and j_0 is not a power of 2. Then at least for all indices $n \geq j_0$ it holds that $a^{j_0} \notin L(G_n)$ since $a^{j_0} \notin \{\varepsilon, a^{2^i} \mid i \in \mathbb{N}\}$ and $2^n + m \cdot 2^n > j_0$ for all $m \in \mathbb{N}$ and $n \geq j_0$. Thus, also $a^{j_0} \notin \limsup_{n\to\infty} L(G_n)$. Consequently, $\limsup_{n\to\infty} L(G_n) \subseteq L(G)$.

(b) If $K_n = \{2^n - 1\}$ and $n \in \mathbb{N}$, then obviously

$$L(G_n) = \{\varepsilon, a, a^{2^i}, a^{2^n + m\cdot(2^n - 1)} \mid i = 1, \ldots, n \wedge m \in \mathbb{N}\}.$$

Because of Remark 6.2 (a) it suffices to show that

$$\limsup_{n\to\infty} L(G_n) \subseteq L(G) \cup \{a\} \subseteq \liminf_{n\to\infty} L(G_n). \tag{6.6}$$

Analogously to the proof of item (a), consider an arbitrary word $a^{j_0} \in \{a\}^* \setminus (L(G) \cup \{a\})$ which means that $j_0 > 1$ and j_0 is not a power of 2.

Then at least for all indices $n \geq j_0$ it holds that $a^{j_0} \notin L(G_n)$ since $a^{j_0} \notin \{\varepsilon, a, a^{2^i} \mid i \in \mathbb{N}\}$ and $2^n + m \cdot (2^n - 1) > j_0$ for all $m \in \mathbb{N}$ and $n \geq j_0$. Thus, also $a^{j_0} \notin \limsup_{n\to\infty} L(G_n)$. Consequently, $\limsup_{n\to\infty} L(G_n) \subseteq L(G) \cup \{a\}$ which proves the first inclusion of (6.6). The second inclusion follows from Theorem 6.4 and the fact that $a \in L(G_n)$ for all $n \in \mathbb{N}$.

(c) Consider the two sub-sequences $(L(G_{2n-1}))_{n\in\mathbb{N}}$ and $(L(G_{2n}))_{n\in\mathbb{N}}$ of the sequence $(L(G_n))_{n\in\mathbb{N}}$. Then according to the definition of the sets K_n, $(L(G_{2n-1}))_{n\in\mathbb{N}}$ equals the sequence of languages as considered in item (a) and $(L(G_{2n}))_{n\in\mathbb{N}}$ equals the sequence of languages as considered in item (b). Therefore, on the one hand, from Remark 6.2 (d) and item (a) it follows that

$$\liminf_{n\to\infty} L(G_n) \subseteq \liminf_{n\to\infty} L(G_{2n-1}) = L(G).$$

On the other hand, from Remark 6.2 (d) and item (b) it follows that

$$\limsup_{n\to\infty} L(G_n) \supseteq \limsup_{n\to\infty} L(G_{2n}) = L(G) \cup \{a\}.$$

Thus, $\liminf_{n\to\infty} L(G_n) \neq \limsup_{n\to\infty} L(G_n)$ and $\lim_{n\to\infty} L(G_n)$ does not exist. ■

Example 6.10 Consider the non-propagating and non-deterministic 0L system $G = (\{a\}, h, a^2)$ with $h(a) = \{\varepsilon, a^2\}$, generating the language $L(G) = \{a^{2i} \mid i \in \mathbb{N}_0\}$, obviously. Furthermore, consider for $n \in \mathbb{N}$ respective $K_n$ml0L systems $G_n = (G, \kappa_n)$ with $\lim_{n\to\infty} \min K_n = \infty$. Then the following holds:

(a) $\lim_{n\to\infty} L(G_n) = L(G)$ if $K_n = \{2n\}$.
(b) $\lim_{n\to\infty} L(G_n) = \{a\}^*$ if $K_n = \{2n-1\}$.
(c) $\lim_{n\to\infty} L(G_n)$ does not exist if $K_n = \{n\}$.

Proof:

(a) If $K_n = \{2n\}$ and $n \in \mathbb{N}$, then obviously $L(G_n) = \{a^{2i} \mid i \in \mathbb{N}_0\} = L(G)$ and thus, $\lim_{n\to\infty} L(G_n) = L(G)$.

(b) If $K_n = \{2n-1\}$ and $n \in \mathbb{N}$, then obviously $L(G_n) = \{a\}^*$ and thus, $\lim_{n\to\infty} L(G_n) = \{a\}^*$.

(c) This item is proved analogously to Example 6.9 (c). Consider the two sub-sequences $(L(G_{2n-1}))_{n\in\mathbb{N}}$ and $(L(G_{2n}))_{n\in\mathbb{N}}$ of the sequence $(L(G_n))_{n\in\mathbb{N}}$. Then according to the definition of the sets K_n, $(L(G_{2n-1}))_{n\in\mathbb{N}}$ equals the sequence of languages as considered in item (a) and $(L(G_{2n}))_{n\in\mathbb{N}}$ equals the sequence

of languages as considered in item (b). Therefore, on the one hand, from Remark 6.2 (d) and item (a) it follows that

$$\liminf_{n\to\infty} L(G_n) \subseteq \liminf_{n\to\infty} L(G_{2n-1}) = L(G).$$

On the other hand, from Remark 6.2 (d) and item (b) it follows that

$$\limsup_{n\to\infty} L(G_n) \supseteq \limsup_{n\to\infty} L(G_{2n}) = \{a\}^*.$$

Thus, $\liminf_{n\to\infty} L(G_n) \neq \limsup_{n\to\infty} L(G_n)$ and $\lim_{n\to\infty} L(G_n)$ does not exist. ■

Chapter 7

Summary and Suggestions for further Research

This chapter summarizes in Section 7.1 the results and open problems of this thesis and provides suggestions for further research in Section 7.2.

7.1 Summary and Open Problems

Multi-limited 0L systems represent a generalization of the k-limited 0L systems which were introduced and investigated by D. Wätjen. They provide a model for plant development where the growth of each cell-type is bounded by an individual reservoir of food, given by a limiting function κ on the alphabet Σ.

In this thesis, the notion of multi-limited 0L systems (κmlET0L systems) was formalized and their generated languages (κmlET0L languages) were classified independently from the respective alphabet. This enabled the definition of the families of KmlET0L languages ($\mathcal{L}(K\text{mlET0L})$) for all non-empty sets $K \subseteq \mathbb{N}$, which consist of all κmlET0L languages with $\text{Im}(\kappa) \subseteq K$.

An intuitive approach to the different mechanisms of the 0L system variants considered in this work was provided by presenting the turtle interpretation as a method for the graphical interpretation of strings. Simultaneously, this outlined an application area of 0L systems outside formal language theory.

The investigation of KmlET0L systems under aspects of formal language theory was the main goal of this thesis. As an result of the investigation of the non-extended case (i.e. multi-limited 0L systems which do not use non-terminal symbols), different sets K always lead to different families $\mathcal{L}(K\text{mlT0L})$. Thus, for $|K| \geq 2$, they all differ from any family of non-extended k-limited 0L languages. Furthermore, the set K even characterizes the families $\mathcal{L}(K\text{ml(P)(D)(T)0L})$ which means that the inclusion relation between two sets K_1, K_2 also holds for the corresponding families $\mathcal{L}(K_1\text{ml(P)(D)(T)0L})$, $\mathcal{L}(K_2\text{ml(P)(D)(T)0L})$ of the same type. Nevertheless, a few inclusion relations between the families $\mathcal{L}(K\text{ml(P)(D)(T)0L})$ remain open.

The families of non-limited T0L languages as well as the families, $\mathcal{L}(\text{fin})$, $\mathcal{L}(\text{reg})$, and $\mathcal{L}(\text{cf})$, of the Chomsky Hierarchy are incomparable with the families $\mathcal{L}(K\text{ml(P)(D)(T)0L})$. It remains open whether this also holds for $\mathcal{L}(\text{cs})$. Finally, regarding closure properties it revealed that the families $\mathcal{L}(K\text{ml(P)(D)(T)0L})$ are anti-AFLs and additionally not closed with respect to concatenation.

In the case of the extended multi-limited 0L systems, the existence of a normal form for KmlE(P)(D)T0L systems as well as a weak iteration theorem concerning KmlE0L systems was proved analogously to the k-limited case. In addition, for KmlE(D)T0L systems it could be demonstrated that the family $\mathcal{L}(K\text{mlE(D)T0L})$ does not grow if K is replaced by a set of finite sums of elements of K. As a consequence of this, the family of all 1lE(D)T0L languages includes all multi-limited ET0L languages. It remains open whether this result also is valid for the case of 1lEPT0L languages.

The properties of the families $\mathcal{L}(K\text{mlE(P)(D)T0L})$ were proved to be analogous to the k-limited case regarding inclusions relations with respect to families of non-limited E(P)(D)T0L languages and the Chomsky Hierarchy, as well as regarding closure and decidability properties. The inclusion relation with $\mathcal{L}(\text{cs})$ as well as the closure property with respect to intersection with regular languages remains open. If this closure property would be given, $\mathcal{L}(K\text{mlET0L})$ would be a full AFL.

Finally, the generative power of KmlET0L systems was compared asymptotically to the underlying non-limited ET0L system for the case that the smallest permitted limit $\min K$ tends to infinity. It turned out that the generative power of an KmlED0L system as well as of an KmlEPT0L system always converges to the generative power of the underlying non-limited system. For all the remaining cases counterexamples were presented.

7.2 Suggestions for further Research

Multi-limited ET0L systems are motivated by biological cell growth processes where the growth of every cell-type is bounded by an individual limit. An extended and, with respect to the simulation of biological phenomena, an even more realistic approach could be the simulation and investigation of *multi-partition-limited 0L systems* as the combination of the assignment of multiple limits and the partitioning of the alphabet Σ of cell-types according to [12]. To every element p of the partition of Σ, a common limit $\kappa(p)$ is assigned. Then, in every derivation step applied to a word w exactly $\min\{k(p), \#_p w\}$ occurrences of symbols $a \in p$ within w are rewritten.

The biological interpretation of this mechanism is a kind of plant development where groups of cell-types have a common finite reservoir of food. With respect to formal language theory, these multi-partition-limited 0L systems represent a simultaneous generalization of k-limited, uniformly k-limited, partition-limited, and multi-limited 0L systems. In the sequel, these systems are formalized and examples of their working method are given.

Definition 7.1 Consider an alphabet Σ and a *partition* P of Σ, i.e. $P = \{p_1, \ldots, p_r\}$ with $1 \leq r \leq |\Sigma|$ and non-empty disjoint subsets $p_1, \ldots, p_r \subseteq \Sigma$ with $\bigcup_{j=1}^{r} p_j = \Sigma$. Furthermore, consider a limiting function $\kappa : P \to \mathbb{N}$ and a finite non-empty substitution h on Σ. Then the mapping $h_{\kappa,P} : \Sigma^* \to \wp(\Sigma^*)$ is called the (κ, P)-*limitation of* h, where $h_{\kappa,P}(w)$ is defined as the set of all words which arise from the word w by simultaneously rewriting all, but at most $\kappa(p)$ occurrences of symbols $a \in p$, for $p \in P$, by an arbitrary word $v \in h(a)$.

For $w = a_1 \cdots a_n$ with $a_i \in \Sigma$ for $i = 1, \ldots, n$ and $n \geq 0$, and $I_p(w) = \{j \mid a_i \in p\}$ for all $p \in P$, this definition also is given by the formula

$$h_{\kappa,P}(w) = \{v_1 \cdots v_n \mid \forall_{p \in P} \quad \exists_{R_p \subseteq I_p(w)} \\ (|R_p| = \min\{\#_p w,\, \kappa(p)\} \wedge \forall_{i \in R_p} v_i \in h(a_i) \wedge \forall_{i \in I_p(w) \setminus R_p} v_i = a_i)\}. \blacksquare$$

Definition 7.2 An ordered sextuplet $G = (\Sigma, H, \omega, \Delta, \kappa, P)$ is called a (κ, P)-*multi-partition-limited ET0L system*, or briefly (κ, P)*mplET0L system* or just *mplET0L system*, if $G' = (\Sigma, H, \omega, \Delta)$ is an ET0L system, $\kappa : \Sigma \to \mathbb{N}$ is a limiting function, and P is a partition of Σ. For each type $\tau \in \mathrm{TYPE}_{\mathrm{ET0L}}$, the system G is called a (κ, P)*mpl*τ*0L* system if G' is a τ0L system. $\blacksquare$

Definition 7.3 Given a (κ, P)mplET0L system $G = (\Sigma, H, \omega, \Delta, \kappa, P)$, the binary *yield relation* $\Longrightarrow_G$ on the set Σ^* is defined as follows: $v \Longrightarrow_G w$ holds if there exists a table $h \in H$ such that $w \in h_{\kappa,P}(v)$. This statement also is written as $v \overset{h}{\Longrightarrow}_G w$ or just as $v \Longrightarrow w$, if it is unambiguous. The transitive or reflexive transitive closure of $\Longrightarrow_G$ is denoted by $\Longrightarrow_G^+$ or $\Longrightarrow_G^*$, respectively. The *language* $L(G)$ *generated by* G is defined as

$$L(G) = E(G) \cap \Delta^* \qquad \text{where} \qquad E(G) = \{w \mid \omega \Longrightarrow_G^* w\}. \blacksquare$$

Example 7.4 (a) Every k-limited ET0L system $G = (\Sigma, H, \omega, \Delta, k)$ can be written as the equivalent (κ, P)mplET0L system $G' = (\Sigma, H, \omega, \Delta, \kappa, P)$ with $P = \{\{a\} \mid a \in \Sigma\}$ and $\kappa(\{a\}) = k$ for all $a \in \Sigma$.

(b) Every uniformly k-limited ET0L system $G = (\Sigma, H, \omega, \Delta, k)$ can be written as the equivalent (κ, P)mplET0L system $G' = (\Sigma, H, \omega, \Delta, \kappa, P)$ with $P = \{\Sigma\}$ and $\kappa(\Sigma) = k$.

(c) Every κ-multi-limited ET0L system $G = (\Sigma, H, \omega, \Delta, \kappa)$ can be written as the equivalent (κ', P)mplET0L system $G' = (\Sigma, H, \omega, \Delta, \kappa', P)$ with $P = \{\{a\} \mid a \in \Sigma\}$ and $\kappa'(\{a\}) = \kappa(a)$ for all $a \in \Sigma$.

(d) The (κ, P)mplPD0L system $G = (\{a, b, c\}, h, abc, \kappa, P)$ with $h(a) = a^2$, $h(b) = b^2$, $h(c) = c^3$, the partition $P = \{\{a, b\}, \{c\}\}$ of Σ, and $\kappa(\{a, b\}) = 3$, $\kappa(\{c\}) = 8$, generates the language

$$\begin{aligned} L = {} & \{abc,\ a^2b^2c^3,\ a^4b^3c^9,\ a^3b^4c^9\} \cup \\ & \{a^pb^qc^{9+16n} \mid p + q = 7 + 3n,\ p, q \geq 3,\ p, q, n \in \mathbb{N}\}. \blacksquare \end{aligned}$$

Appendix A

Source Code

The examples presented in Chapter 1 and Chapter 3, regarding strings generated by 0L systems as well as regarding their turtle interpretations, were produced with the help of special C++ computer programs which were developed within the frame of this thesis. The implementation of these examples are given in Section A.1. The respective source-code of a turtle interpreter as well as the source-code of free-programmable simulators for multi-limited, k-limited, and uniformly k-limited 0L systems can be found in the sections A.3–A.5. Sections A.6 and A.7 contain the source code of commonly used auxiliary classes.

A.1 Examples of Turtle Interpretations

For each example of a turtle interpretation presented in this work, the following subsections list the system defining input data for the programmable simulator for the respective limited 0L system as well as the individual angle increment δ and the command subsets, Σ_F and Σ_f, used by the turtle interpreter.

A.1.1 Tree-like Branching Structures

The turtle interpretation of the tree-like branching structures presented in Figure 1.2, page 4, Figure 1.3, page 5, Figure 3.3, page 35, and Figure 3.4, page 36, was produced using the bracketed (uniformly) k-limited 0L system

```
    k = 100;
omega = A;
    p = A -> CWLB;
    p = B -> CWRA;
    p = L -> [+A];
    p = R -> [-B];
    p = W -> X;
    p = X -> WC;
```

respectively the bracketed multi-limited 0L system

```
    k = A -> 10;
    k = B -> 100;
    k = L -> 100;
    k = R -> 100;
    k = W -> 100;
    k = X -> 100;
omega = A;
    p = A -> CWLB;
    p = B -> CWRA;
    p = L -> [+A];
    p = R -> [-B];
    p = W -> X;
    p = X -> WC;
```

and the bracketed turtle $T_b = (\Sigma \cup \{[,]\}, Z \times S, d, \delta)$ with the angle increment $\delta = 30°$, $\Sigma_F = \{a, b, c, l, r, w, x\}$, and $\Sigma_f = \emptyset$.

A.1.2 Hexagonal Gosper Curves

The turtle interpretation of the FASS curves named "hexagonal Gosper" curves presented in Figure 3.1, page 32, was produced using the k-limited 0L system, $k =$ 1.000.000,

```
omega= L;
```

```
    p= L->L+R++R-L--LL-R+;
    p= R->-L+RR++R+L--L-R;
```

and the turtle $T = (\Sigma, Z, d, \delta)$ with the angle increment $\delta = 60°$, $\Sigma_F = \{L, R\}$, and $\Sigma_f = \emptyset$.

A.1.3 Combination of islands and lakes

The turtle interpretation of the quadratic Koch curves named "Combination of islands and lakes" presented in Figure 3.2, page 33, was produced using the k-limited 0L system, $k = 1.000.000$,

```
omega= F+F+F+F;
    p= F->F+f-FF+F+FF+Ff+FF-f+FF-F-FF-Ff-FFF;
    p= f->ffffff;
```

and the turtle $T = (\Sigma, Z, d, \delta)$ with angle increment $\delta = 90°$, $\Sigma_F = \{F\}$, and $\Sigma_f = \{f\}$.

A.2 Bracketed Turtle Interpreter

```
/*
    Author: Markus Seemann
    Date:   2007-07-07
*/
#include <iostream.h>
#include <fstream.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <syslimits.h>

#include "DListTemp.h"
#include "mathe.h"

const char LF = '\n';
const char CR = '\r';
const unsigned short int  maxWordLen = USHRT_MAX;

typedef char*        TFileName;
typedef char         TWord[maxWordLen];
```

```
typedef double       TAngle;
typedef double       TWidth;
typedef double       Tcoor;
typedef stringList   TStrList;

TFileName  infile = "", outfile = "outfile.ps";
TWidth     width  = 0.1;    // real line width in pt.
TWord      buffer = "";
TAngle     delta  = 30;     // degrees of turning angle
TStrList   wordList;        // entered strings of turtle commands

class Point {
    public:

    Tcoor x,y;

    Point(Tcoor a = 0, Tcoor b = 0)
    {
        x = a; y = b;
    };
};

unsigned int operator ==(Point a, Point b)
{
    return ( (a.x == b.x) && (a.y == b.y) );
}

class Box {
    public:

    Point downleft,upright;

    Box (Point dl = Point(0,0), Point ur = Point(0,0))
    {
        downleft = dl; upright = ur;
    };

};

unsigned int operator ==(Box a, Box b)
{
    return ( (a.downleft == b.downleft) && (a.upright == b.upright) );
}

class State {
    public:

    Point  pos;
    TAngle alpha;

    State (Point p = Point(0,0), TAngle a = 0)
    {
        pos = p; alpha = a;
    };
```

```
};

class Turtle {
    public:

    State state;
    Tcoor  dist;      // distance
    TAngle delta;     // turning angle

    Turtle(State s, Tcoor  d, TAngle a)
    {
        state = s; dist = d; delta = a;
    };
};

const Point        maxPoint  = Point(610,790);// upper right corner.
const TAngle       startDir  = 90;            // starting direction in degrees.
const Tcoor        startDist = 1;             // starting distance.
const unsigned int prec      = 5;             // precision of output.

Box    boundingBox (Point(0,0),Point(0,0));
Point  scale       (1,1);
Tcoor  margin      = 0;            // margin in pt.
Tcoor  gap         = 10;            // gap between two graphics in pt.
Tcoor  unit        = 1;             // unit length in pt.

unsigned int AUTOSCALE = 1;  // 0 iff unit is entered.
unsigned int GROUND = 1;  // 0 iff the ground line shall be suppressed.
unsigned int InitialArrow = 1;  // 0 iff the initial arrow shall be suppressed.
unsigned int EndArrow = 1;      // 0 iff the ending arrow shall be suppressed.


static void usage(const char *progname)
// Print a message describing program and options
{
    cerr << progname << " converts a string of turtle commands into a postscript grafic.\n"
         << "Usage: " << progname
         << " [-h] [-noground] [-noInitialArrow] [-noEndArrow] [-i file] [-o file] [-d angle]"
         << "[-w width] [-m margin] [-g gap] [-u unit] [word .. word]\n"
 << "-h               prints this page\n"
 << "-noground        suppresses the ground line\n"
 << "-noInitialArrow  suppresses the initial arrow\n"
 << "-noEndArrow      suppresses the ending arrow\n"
 << "-i         specifies input file, e.g. (in arbitrary order):\n"
 << "              delta  = 42;\n"
 << "              width  = 0.2;\n"
 << "              margin = 20;\n"
 << "              gap    = 10;\n"
 << "              word1  = +gOo[+fi]-E;\n"
 << "              word2  = baba;\n"
 << "-o         specifies output file (default '" << outfile << "')\n"
 << "-d         sets default degrees of turning angle delta (default " << delta << ")\n"
 << "-w         sets default line width in pt (default " << width << " pt)\n"
 << "-m         sets margin in pt (default " << margin << " pt)\n"
```

```
 << "-g         sets gap in pt (default " << gap << " pt)\n"
 << "-u         sets unit length in pt (default: autoscale)\n"
 << "word       string of turtle commands:\n"
 << "           A,..,Z : go straight forward 1 unit drawing\n"
 << "           a,..,z : go straight forward 1 unit without drawing\n"
 << "                + : turn left\n"
 << "                - : turn right\n"
 << "                [ : begin of branch\n"
 << "                ] : end of branch\n"
 << "            other : ignore\n";
}

void error(const int n)
{
    cerr << "***Error(" << n << "): ";
    switch (n) {
case  0: cerr << "No turtle command specified!\n";
         break;
case  1: cerr << "Can't open file " << infile << "\n";
         break;
case  2: cerr << "Syntax error in file " << infile << "\n";
         break;
case  3: cerr << "Can't open file " << outfile << "\n";
         break;
case  4: cerr << "Writing error at file " << outfile << "\n";
         break;
default: break;
    }
}

////////////////////// readinput vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

int Getline(ifstream &in, char* str, int size, const char sep = LF)
// Like getline, but ignores white spaces. Returns 0 iff not ok.
{
    while (1) {
        char c = in.peek();
        if ((c == ' ') || (c == LF) || (c == CR) || (c == '\t')) {
            in.get(str[0]);
        } else {
            break;
        }
    }
    return(!(in.getline(str,size,sep)).eof());
}

int defParam(ifstream &in, char* str, int size)
// Returns 0 iff no correct definition found.
{
    const char*  keyword[] = {"delta","width","margin","gap","unit","word"};
    const int    keyno = 6;

    for (int i = 0; i < keyno; i++) {
        if (!strncmp(str,keyword[i],strlen(keyword[i]))) {           // keyword[i] recognized
```

```
            switch (i) {
case  0: // delta
         if ( (in >> delta) &&
              Getline(in,str,size,';') &&
              (strlen(str) == 0) ) {
     return 1;
 }
 break;
case  1: // width
         if ( (in >> width) &&
              Getline(in,str,size,';') &&
              (strlen(str) == 0) ) {
     return 1;
 }
          break;
case  2: // margin
         if ( (in >> margin) &&
              Getline(in,str,size,';') &&
              (strlen(str) == 0) ) {
     return 1;
 }
          break;
case  3: // gap
         if ( (in >> gap) &&
              Getline(in,str,size,';') &&
              (strlen(str) == 0) ) {
     return 1;
 }
          break;
case  4: // unit
         if ( (in >> unit) &&
              Getline(in,str,size,';') &&
              (strlen(str) == 0) ) {
     return 1;
 }
          break;
case  5: // word
         if ( Getline(in,buffer,sizeof(buffer),';') &&
              (strlen(buffer) > 0) ) {
             wordList.addEnd(buffer);
     return 1;
 }
          break;
default: break;
    }
            error(2);
            return 0;
        }
    }
    error(2);
    return 0;
}

int parse()
```

```
/* Accepted syntax:  (in arbitrary order)
                     delta = 42;
                     width = 0.2;
                     margin = 20;
                     gap    = 10;
                     unit   = 15;
                     word1 = +gOo[+fi]-E;
                     word2 = baba;
   Returns 0 iff not ok.
*/
{
    ifstream in(infile);
    char     str[128];

    if (!in) {
        error(1);
        return 0;
    }
    while (Getline(in,str,sizeof(str),'=')) {
        if (!defParam(in,str,sizeof(str))) {
            return 0;
        }
    }
    if (strlen(str) > 0) {
        error(2);
        return 0;
    } else {
        return 1;
    }
}

void readinput(int argc, char *argv[])
// Reads input from command line.
{
    for (int i = 1; i < argc; i++) {
    if (!strcmp(argv[i], "-h")) {
              usage(argv[0]);
              exit(0);
    } else if (!strcmp(argv[i], "-noground")) {
        GROUND = 0;
    } else if (!strcmp(argv[i], "-noInitialArrow")) {
        InitialArrow = 0;
    } else if (!strcmp(argv[i], "-noEndArrow")) {
        EndArrow = 0;
    } else if (!strcmp(argv[i], "-i")) {
        infile = argv[++i];
        if (!parse())
            exit(0);
    } else if (!strcmp(argv[i], "-o")) {
        outfile = argv[++i];
    } else if (!strcmp(argv[i], "-d")) {
        delta = atof(argv[++i]);
    } else if (!strcmp(argv[i], "-w")) {
        width = atof(argv[++i]);
```

```
    } else if (!strcmp(argv[i], "-m")) {
        margin = atof(argv[++i]);
    } else if (!strcmp(argv[i], "-g")) {
        gap = atof(argv[++i]);
    } else if (!strcmp(argv[i], "-u")) {
        AUTOSCALE = 0;
        unit = atof(argv[++i]);
    } else {
        wordList.addEnd(argv[i]);
    }
    }
    if (wordList.number() == 0){
     error(0);
     usage(argv[0]);
     exit(0);
    }
}

////////////////////// readinput ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

////////////////////// generatePS vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

void printHeader(ofstream &out)
// Prints PS-header to file out.
{
    out << "%!PS-Adobe-2.0 EPSF-2.0\n%%BoundingBox: ";
    out.width(prec+10); out << boundingBox.downleft.x;
    out.width(prec+10); out << boundingBox.downleft.y;
    out.width(prec+10); out << boundingBox.upright.x;
    out.width(prec+10); out << boundingBox.upright.y << LF
        << "%!Device scaling parameters\n";
//     out.width(prec+5); out << margin;
//     out.width(prec+5); out << margin;
//     out << " translate\n";
    out.width(prec+5); out << scale.x;
    out.width(prec+5); out << scale.y;
    out << " scale\n"
        << "%!\n"
        << "/turtlePS_saveobj save def\n"
        << "1 setlinecap\n"
        << "/m { moveto } bind def\n"
        << "/l { lineto } bind def\n"
        << "/w { setlinewidth } bind def\n"
        << "/c { setgray } bind def\n"
        << "/s { stroke } bind def\n"
        << "0 c\n";
    out.width(prec+5); out << width;
    out << " w\n"
        << "newpath" << endl;
    if (!out){
        error(4);
        exit(0);
    }
    return;
```

```
}

////////////////////// printTurtle vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

void printTo(ofstream &out, Point &pos, char c)
// Prints the action to out.
{
    out << pos.x << ' ' << pos.y << "   " << c << endl;
    if (!out){
        error(4);
        exit(0);
    }
    return;
}

void printMoveTo(ofstream &out, Point pos)
// Prints the move-to-action to out.
{
    printTo(out,pos,'m');
    return;
}

void printLineTo(ofstream &out, Point pos)
// Prints the line-to-action to out.
{
    printTo(out,pos,'l');
    return;
}

TAngle rad(TAngle a)
{
    return (a * M_PI / 180);
}

TAngle deg(TAngle a)
{
    return (a * 180 / M_PI);
}

void updateBBox(Turtle &turtle, Box &BBox)
// Derives new position after one step forward; updates the boundingBox.
{
    BBox.downleft.x = min(turtle.state.pos.x, BBox.downleft.x);
    BBox.downleft.y = min(turtle.state.pos.y, BBox.downleft.y);
    BBox.upright.x = max(turtle.state.pos.x, BBox.upright.x);
    BBox.upright.y = max(turtle.state.pos.y, BBox.upright.y);

    return;
}

void printArrow(ofstream &out, double ar_unit, Turtle turtle)
// Prints initial state arrow.
{
    Tcoor old_pos_x = turtle.state.pos.x;
```

```
    Tcoor old_pos_y = turtle.state.pos.y;
    TAngle old_angle = turtle.state.alpha;
    double q = 3;
    double beta = deg(acos(1/q));
    turtle.state.alpha -= rad(90);
    turtle.state.pos.x += turtle.dist * cos(turtle.state.alpha) * ar_unit / q;
    turtle.state.pos.y += turtle.dist * sin(turtle.state.alpha) * ar_unit / q;
    printLineTo(out,turtle.state.pos);
    turtle.state.alpha += rad(180-beta);
    turtle.state.pos.x += turtle.dist * cos(turtle.state.alpha) * ar_unit;
    turtle.state.pos.y += turtle.dist * sin(turtle.state.alpha) * ar_unit;
    printLineTo(out,turtle.state.pos);
    turtle.state.alpha += rad(2*beta);
    turtle.state.pos.x += turtle.dist * cos(turtle.state.alpha) * ar_unit;
    turtle.state.pos.y += turtle.dist * sin(turtle.state.alpha) * ar_unit;
    printLineTo(out,turtle.state.pos);
    turtle.state.alpha += rad(180-beta);
    turtle.state.pos.x += turtle.dist * cos(turtle.state.alpha) * ar_unit / q;
    turtle.state.pos.y += turtle.dist * sin(turtle.state.alpha) * ar_unit / q;
    printLineTo(out,turtle.state.pos);
    turtle.state.pos.x = old_pos_x;
    turtle.state.pos.y = old_pos_y;
    turtle.state.alpha = old_angle;
    printMoveTo(out,turtle.state.pos);
}

unsigned int continuePrintAt(ofstream &out, unsigned int start, char * word, Turtle turtle)
// Interprets the command word beginning at index start.
{
    unsigned int i;
    char wi;

    for (i = start; i < strlen(word); i++) {
        wi = word[i];
        if (('A' <= wi) && (wi <= 'Z')) {
            turtle.state.pos.x += turtle.dist * cos(turtle.state.alpha);
            turtle.state.pos.y += turtle.dist * sin(turtle.state.alpha);
            printLineTo(out,turtle.state.pos);
        } else if (('a' <= wi) && (wi <= 'z')) {
            turtle.state.pos.x += turtle.dist * cos(turtle.state.alpha);
            turtle.state.pos.y += turtle.dist * sin(turtle.state.alpha);
            printMoveTo(out,turtle.state.pos);
        } else if (wi == '+') {
            turtle.state.alpha += turtle.delta;
        } else if (wi == '-') {
            turtle.state.alpha -= turtle.delta;
        } else if (wi == '[') {
            i = continuePrintAt(out,i+1,word,turtle);
            printMoveTo(out,turtle.state.pos);
        } else if (wi == ']') {
            return i;
        } else {
            cerr << "ignored symbol: " << word[i] << endl;
        };
```

```
    };
    if (EndArrow) {
printArrow(out,0.75,turtle);
        printArrow(out,0.50,turtle);
        printArrow(out,0.25,turtle);
    };
    return i;
}

unsigned int continueAt(unsigned int start, char * word, Turtle turtle, Box &BBox)
// Interprets the command word beginning at index start; updates the bounding box BBox.
{
    unsigned int i;
    char wi;

    for (i = start; i < strlen(word); i++) {
        wi = word[i];
        if ((('A' <= wi) && (wi <= 'Z')) || (('a' <= wi) && (wi <= 'z'))) {
            turtle.state.pos.x += turtle.dist * cos(turtle.state.alpha);
            turtle.state.pos.y += turtle.dist * sin(turtle.state.alpha);
            updateBBox(turtle,BBox);
        } else if (wi == '+') {
            turtle.state.alpha += turtle.delta;
        } else if (wi == '-') {
            turtle.state.alpha -= turtle.delta;
        } else if (wi == '[') {
            i = continueAt(i+1,word,turtle,BBox);
        } else if (wi == ']') {
            return i;
        }
    }
    return i;
}

Point newStartPos(char *word)
// Returns the starting position of the turtle and derives the bounding box of the whole graphic.
{
    Turtle turtle(State(Point(0,0),rad(startDir)),startDist,rad(delta));
    Box    BBox  (Point(0,0),Point(0,0));

    continueAt(0,word,turtle,BBox);
    if ( !(boundingBox == Box(Point(0,0),Point(0,0))) ) {
        boundingBox.upright.x += gap;
    }
    boundingBox.upright.x += BBox.upright.x - BBox.downleft.x;
    boundingBox.upright.y = max( BBox.upright.y, boundingBox.upright.y );
    boundingBox.downleft.y = min( BBox.downleft.y, boundingBox.downleft.y );

      // align right, starting position at height 0.
    return Point( boundingBox.upright.x - BBox.upright.x, 0 );
}

void printTurtle(ofstream &out)
// Prints all turtle graphics to out.
```

```
{
    Turtle turtle(State(Point(0,0),rad(startDir)),startDist,rad(delta));
    char * word;

    for (unsigned int i = 1; i <= wordList.number(); i++) {
        cout << "word " << i << " ... ";
        word = wordList.data(i);
        turtle.state.pos = newStartPos(word);
        printMoveTo(out,turtle.state.pos);
if (InitialArrow) {
    printArrow(out,0.75,turtle);
    printArrow(out,0.50,turtle);
    printArrow(out,0.25,turtle);
};
        continuePrintAt(out,0,word,turtle);
        cout << "done." << endl;
    }
    if (GROUND) {
        printMoveTo(out,0);
        printLineTo(out,Point( boundingBox.upright.x, 0 ));
    }

    return;
}

////////////////////////// printTurtle ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

void printTail(ofstream &out)
{
    out << "stroke\n"
        << "turtlePS_saveobj restore\n"
        << "showpage\n";
    if (!out){
        error(4);
        exit(0);
    }
    return;
}

void calibrate()
{
    Tcoor f = unit;

    boundingBox.upright.x += margin;
    boundingBox.upright.y += margin;
    boundingBox.downleft.x -= margin;
    boundingBox.downleft.y -= margin;

    if (AUTOSCALE) {
        f = min(maxPoint.x / (boundingBox.upright.x - boundingBox.downleft.x),
                maxPoint.y / (boundingBox.upright.y - boundingBox.downleft.y));
    }
    scale.x = f;
    scale.y = f;
```

```
    boundingBox.upright.x *= f;
    boundingBox.upright.y *= f;
    boundingBox.downleft.x *= f;
    boundingBox.downleft.y *= f;

//     width *= f;

    return;
}

void generatePS()
{
    ofstream out(outfile);

    if (!out) {
        error(3);
        exit(0);
    }
    out.precision(prec);
    out.setf(ios::fixed);
    printHeader(out);
    printTurtle(out);
    printTail(out);
    calibrate();
    out.seekp(streampos(0));
    printHeader(out);

    return;
}

////////////////////// generatePS ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

int main(int argc, char *argv[])
{
    readinput(argc,argv);

    cout << "infile  = " << infile << "\n";
    cout << "outfile = " << outfile << "\n";
    cout << "delta   = " << delta << " degrees\n";
    cout << "width   = " << width << " pt\n";
    cout << "margin  = " << margin << " pt\n";
    cout << "gap     = " << gap << " pt\n";
    if (AUTOSCALE)
        cout << ", autoscaled with image.\n";
    else
        cout << "\nunit    = " << unit << " pt" << endl;
    for (unsigned int i = 1; i <= wordList.number(); i++) {
        cout << "word" << i <<" = " << wordList.data(i) << "\n";
    }

    generatePS();
}
```

A.3 Simulator for ml0L systems

```
/*
    Author: Markus Seemann
    Date:   2007-07-07
*/
#include <iostream.h>
#include <fstream.h>
#include <string.h>
#include <stdlib.h>
#include <syslimits.h>

#include "DListTemp.h"
#include "mathe.h"

const char LF = '\n';
const char CR = '\r';
const unsigned short int  maxWordLen     = USHRT_MAX;
const unsigned int        maxProdWordLen = 50;

typedef char*               TFileName;
typedef char                TWord[maxWordLen];
typedef char                TProdWord[maxProdWordLen];
typedef unsigned int        TLimit[256];
typedef stringList          Table;
typedef stringList          TStrList;

class LSystem {
    public:

    TLimit limit;
    char*  omega;
    Table  table[256];

    LSystem(char * s = "")
    {
        for (unsigned int i = 0; i < 256; i++)
            limit[i] = 0;
        omega = s;
    }
};

unsigned int TERMINAL = 0;    // 0 iff all generated words shall be printed.
unsigned int ALLDERIV = 0;    // 0 iff only distinct words shall be generated.
unsigned int RANDOM   = 0;    // 0 iff all words shall be generated randomly.

TFileName    infile = "", outfile = "outfile.wrd";
unsigned int step  = 1;       // number of derivation steps.
TWord        buffer = "";
LSystem      lsystem(buffer);
```

```
void copy(char *s, char *t)
// Copies strlen(s)+1 (incl. final '\0') characters from s to t.
{
    for (unsigned int i = 0; i < strlen(s)+1; i++)
        t[i] = s[i];
}

static void usage(const char *progname)
// Print a message describing program and options
{
    cerr << progname << " prints all words generated by a klET0L-system in n derivation steps.\n"
         << "Usage: " << progname <<
            " [-h] [-terminal] [-all] [-random] [-i file] [-o file] [-k A limit] [-omega axiom]" <<
            " [-p A P] [-step n]\n"
 << "-h           prints this page\n"
 << "-terminal    only words without symbols A,..,Z are written to output file\n"
 << "-all         all derivable words are generated\n"
 << "-random      from all derivable words in a step one is selected randomly\n"
 << "-i           specifies input file, e.g. (in arbitrary order):\n"
 << "                    k = A->5;\n"
 << "                     ...\n"
 << "                    k = b->42;\n"
 << "                omega = hello;\n"
 << "                    p = A->cAb;\n"
 << "                     ...\n"
 << "                    p = b->c;\n"
 << "-o           specifies output file (default '" << outfile << "')\n"
 << "-k           specifies the limit of the symbol A (default limit = 0)\n"
 << "-omega       specifies the axiom of the system\n"
 << "-p           specifies a production A P of the system\n"
 << "             where A is a symbol and P is a sequence of symbols\n"
 << "-step        number of derivation steps (default n = 1).\n";
}

void error(const int n)
{
    cerr << "***Error(" << n << "): ";
    switch (n) {
case  0: cerr << "No axiom specified!\n";
         break;
case  1: cerr << "Can't open file " << infile << "\n";
         break;
case  2: cerr << "Syntax error in file " << infile << "\n";
         break;
case  3: cerr << "Can't open file " << outfile << "\n";
         break;
case  4: cerr << "Writing error at file " << outfile << "\n";
         break;
case  5: cerr << "Syntax error in parameter specification:\n";
         break;
case  6: cerr << "Wrong number selected.\n";
         break;
default: break;
    }
```

```
}

////////////////////// readinput vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

int Getline(ifstream &in, char* str, int size, const char sep = LF)
// Like getline, but ignores white spaces. Returns 0 iff not ok.
{
    while (1) {
        char c = in.peek();
        if ((c == ' ') || (c == LF) || (c == CR) || (c == '\t')) {
            in.get(str[0]);
        } else {
            break;
        }
    }
    return(!(in.getline(str,size,sep)).eof());
}

int isTerminal(char *s)
// Returns 0 iff string s contains a character A,..,Z.
{
    for (unsigned int i = 0; i < strlen(s); i++) {
        if ( (s[i] >= 'A') && (s[i] <= 'Z') )
            return 0;
    }
    return 1;
}

int addProduction(char c, char* s)
// Returns 0 iff not ok.
{
    if (isTerminal(s)) {
        return lsystem.table[c].addBeg(s);
    } else {
        return lsystem.table[c].addEnd(s);
    }
}

int defParam(ifstream &in, char* str, int size)
// Returns 0 iff no correct definition found.
{
    const char*  keyword[] = {"k","omega","p"};
    const int    keyno = 3;
    TProdWord    pw;

    for (int i = 0; i < keyno; i++) {
        if (!strncmp(str,keyword[i],strlen(keyword[i]))) {   // keyword[i] recognized
            switch (i) {
case  0: // k
         if ( Getline(in,str,sizeof(str),'>') &&
              (in >> lsystem.limit[str[0]]) &&
              Getline(in,str,size,';') &&
              (strlen(str) == 0) ) {
     return 1;
```

```
 }
 break;
case  1: // omega
         if ( Getline(in,buffer,sizeof(buffer),';') &&
              (strlen(buffer) > 0) ) {
             lsystem.omega = buffer;
     return 1;
 }
          break;
case  2: // p
         if ( Getline(in,str,sizeof(str),'>') &&
              Getline(in,pw,sizeof(pw),';') ) {
             addProduction(str[0],pw);
     return 1;
           }
          break;
default: break;
    }
           error(2);
           return 0;
        }
    }
    error(2);
    return 0;
}

int parse()
/* Accepted syntax:  (in arbitrary order)
      k = A->5;
       ...
      k = b->42;
  omega = hello;
      p = A->cAb;
       ...
      p = b->c;
   Returns 0 iff not ok.
*/
{
    ifstream in(infile);
    char     str[128];

    if (!in) {
        error(1);
        return 0;
    }
    while (Getline(in,str,sizeof(str),'=')) {
        if (!defParam(in,str,sizeof(str))) {
            return 0;
        }
    }
    if (strlen(str) > 0) {
        error(2);
        return 0;
    } else {
```

```
        return 1;
    }
}

void readinput(int argc, char *argv[])
// Reads input from command line.
{
    for (int i = 1; i < argc; i++) {
if (!strcmp(argv[i], "-h")) {
         usage(argv[0]);
         exit(0);
} else if (!strcmp(argv[i], "-terminal")) {
    TERMINAL = 1;
} else if (!strcmp(argv[i], "-all")) {
    ALLDERIV = 1;
} else if (!strcmp(argv[i], "-random")) {
    RANDOM = 1;
} else if (!strcmp(argv[i], "-i")) {
    infile = argv[++i];
    if (!parse())
        exit(0);
} else if (!strcmp(argv[i], "-o")) {
    outfile = argv[++i];
} else if (!strcmp(argv[i], "-k")) {
    if (i + 1 < argc) {
        lsystem.limit[argv[i + 1][0]] = atoi(argv[i + 2]);
        i += 2;
    } else {
error(5);
usage(argv[0]);
exit(0);
    }
} else if (!strcmp(argv[i], "-omega")) {
    lsystem.omega = argv[++i];
} else if (!strcmp(argv[i], "-p")) {
    if (i + 1 < argc) {
        addProduction(argv[i + 1][0],argv[i + 2]);
        i += 2;
    } else {
error(5);
usage(argv[0]);
exit(0);
    }
} else if (!strcmp(argv[i], "-step")) {
    step = atoi(argv[++i]);
} else {
         usage(argv[0]);
         exit(0);
  }
    }
    if (strlen(lsystem.omega) == 0){
     error(0);
     usage(argv[0]);
     exit(0);
```

```
    }
}

////////////////////// readinput ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

////////////////////// printList vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

void printWord(ofstream &out, char* w, unsigned int s, long int  c)
{
    out << "word" << s << '_' << c << " = " << w << ';' << endl;
    return;
}

void printList(ofstream &out, TStrList l, unsigned int s)
{
    unsigned short int  c = 0;

    if (TERMINAL) {
        for (unsigned short int i = 1; i < l.number() + 1; i++)
            if ( isTerminal(l.data(i)) ) {
                c++;
            }
    } else {
        c = l.number();
    }

//     out << "All " << c << " ";
//     if (TERMINAL) {
//         out << "terminal ";
//     }
//     out << "words generated by " << s << " derivation steps:" << endl;

    c = 0;
    if (TERMINAL) {
        for (unsigned short int i = 1; i < l.number() + 1; i++)
            if ( isTerminal(l.data(i)) ) {
                c++;
                printWord(out,l.data(i),s,c);
                out << endl;
            }
    } else {
        for (unsigned short int i = 1; i < l.number() + 1; i++) {
            printWord(out,l.data(i),s,i);
            out << endl;
        }
    }
    if (!out) {
        error(3);
        exit(0);
    }

    return;
}
```

```
////////////////////// printList ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

////////////////////// derive vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

void deriveWord(char *w, unsigned int prod[], char *v)
// Writes the derivation of w according to prod into the string v.
{
    v[0] = '\0';
    for (unsigned int i = 0; i < strlen(w); i++) {
        if (prod[i] > 0) {
            copy(lsystem.table[w[i]].data(prod[i]), v + strlen(v));
        } else {
            v[strlen(v) + 1] = '\0';
            v[strlen(v)] = w[i];
        }
    }

    return;
}

////////////////////// firstConstellation vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

void firstConstellation(char *w, unsigned int prod[])
{
    unsigned int limitLeft[256];
    // maximum number of occurences in w of every symbol to be rewritten.

    for (int i = 0; i < 256; i++) { limitLeft[i] = lsystem.limit[i]; }

    for (unsigned int i = 0; i < strlen(w); i++) {
        if (limitLeft[w[i]] > 0) {
            limitLeft[w[i]]--;
            prod[i] = lsystem.table[w[i]].number();  // also works if one symbol has 0 productions
        }
    }

    return;
}

////////////////////// firstConstellation ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

////////////////////// nextConstellation vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

int nextSymbolConf(char *w, unsigned int prod[], char c)
// Returns 0 iff there is no further configuration.
{
    unsigned int pos[maxWordLen]; // positions of the occurrences of c in w.
    unsigned int len = 0;
    unsigned int over = 0;

    for (unsigned int i = 0; i < strlen(w); i++) {
        if (w[i] == c) {
            pos[len] = i;
            len++;
```

```
        }
    }

    for (int i = len - 1; i >= 0; i--) {
        if (prod[pos[i]] > 0) {
            over++;
        } else {
            for (int j = i - 1; j >= 0; j--) {
                if (prod[pos[j]] > 0) {        // one further configuration found
                    for (unsigned int k = 0; k < over; k++) {
                        prod[pos[len - k - 1]] = 0;
                    }
                    for (unsigned int k = 0; k < over + 1; k++) {
                        prod[pos[j + k + 1]] = lsystem.table[c].number();
                    }
                    prod[pos[j]] = 0;
                    return 1;
                }
            }
            // no further configuration
            for (unsigned int k = 0; k < over; k++) {
                prod[pos[len - k - 1]] = 0;
            }
            for (unsigned int k = 0; k < over; k++) {
                prod[pos[k]] = lsystem.table[c].number();
            }
            return 0;
        }
    }

    return 0;
}

int nextConf(char *w, unsigned int prod[], unsigned int ix, char pres[], unsigned int &len)
// Returns 0 iff there is no further configuration, i.e. the choice of the
// symbols to be rewritten.
// ix = int(c) where c is the considered limited symbol.
{
    if (nextSymbolConf(w,prod,pres[ix])) {
        return 1;
    }
    if (ix + 1 < len) {
        return nextConf(w,prod,ix + 1,pres,len);
    }
    return 0;
}

int nextConstellation(char *w, unsigned int prod[], char pres[], unsigned int &len)
// Returns 0 iff there is no further constellation, i.e. the choice of the
// symbols to be rewritten and of the productions.
{
    unsigned int m = strlen(w);

    for (unsigned int i = 0; i < m; i++) {
```

```
        if (prod[i] > 1) {
            prod[i]--;
            return 1;
        } else if (prod[i] == 1) {  // overflow -> search next counter
            prod[i] = lsystem.table[w[i]].number();
        }
    }

    // overflow of all counters -> search next configuration
    if (nextConf(w,prod,0,pres,len)) {
        return 1;
    }

    return 0;  // no further derivation found
}

////////////////////// nextConstellation ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

////////////////////// randomConstellation vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

void randomConstellation(char *w, unsigned int prod[],
                         char pres[], unsigned int &len)
// Randomly chooses a constellation.
{
    char a;
    unsigned int wlen = strlen(w);
    unsigned int ix[maxWordLen];  // indizes of remaining occurrences.
    unsigned int ixlast; // index of the last of such indizes.
    unsigned int K;  // number of a's to rewrite.
    unsigned int sel;
    // position of the selected occurrence of a in the vector of unselected occurrences.

    for (unsigned int i = 0; i < len; i++) { // for each cell type a ...
        a = pres[i];
        if (lsystem.table[a].number() > 0) {
            ixlast = 0;
            for (unsigned int j = 0; j < wlen; j++) {    // initialize ix.
                if (w[j] == a) {
                    ix[ixlast] = j;
                    ixlast++;
                }
            }
            K = min(ixlast,lsystem.limit[a]);  // number of a's to rewrite.
            ixlast--;
            for (unsigned int j = 0; j < K; j++) {      // select K times.
                sel = random(ixlast);
                prod[ix[sel]] = random(lsystem.table[a].number() - 1) + 1;
                ix[sel] = ix[ixlast];                   // remove selected index from ix.
                ixlast--;
            } // for j
        } // if
    } // for i

    return;
```

```
}

////////////////////// randomConstellation ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Integer derivNum(unsigned int parikh[], char pres[], unsigned int &len)
// Returns the number of distinct derivations.
{
    Integer r = 1;

    for (unsigned int i = 0; i < len; i++) {
        char a = pres[i];
        unsigned long int num = lsystem.table[a].number();
        if (num > 0) {
            unsigned int m = min( lsystem.limit[a], parikh[a] );
            r *= choose(parikh[a],m) * pow(num,m);
        }
    }

    return r;
}

void deriveWordList(char *w, TStrList &list)
{
    unsigned int prod[maxWordLen]; // number of the production to be applied to a symbol of w.
    unsigned int parikh[256];      // parikh vector of w.
    char         pres[256];        // all symbols c with parikh[c] > 0.
    unsigned int len = 0;   // Length of pres.
    TWord v;
    Integer      num;
    unsigned int sel;

    setRandom(sel);

    if (strlen(w) == 0) {
        list.addEnd(w);
    } else {
        for (unsigned int i = 0; i < maxWordLen; i++) { prod[i] = 0; }
        for (unsigned int i = 0; i < 256; i++) { parikh[i] = 0; }
        for (unsigned int i = 0; i < strlen(w); i++) { parikh[w[i]]++; }
        for (unsigned int i = 0; i < 256; i++) {
            if (parikh[i] > 0) {
                pres[len] = i;
                len++;
            }
        }

        if (ALLDERIV) {
            firstConstellation(w,prod);
            deriveWord(w,prod,v);
            list.addEnd(v);
            while (nextConstellation(w,prod,pres,len)) {
                deriveWord(w,prod,v);
                list.addEnd(v);
            }
```

```
        } else if (RANDOM) {
            cout << "Selecting one out of all derivations ... ";
            randomConstellation(w,prod,pres,len);
            cout << "done." << endl;
            deriveWord(w,prod,v);
            list.addEnd(v);
        } else {
            if ( (num = derivNum(parikh,pres,len)) > 1) {
                cout << "Enter an arbitrary number [0.." << LONG_MAX << "] : ";
                cin >> sel;
                setRandom(sel);
                cout << "Selecting one out of " << num << " derivations ... ";
                randomConstellation(w,prod,pres,len);
                cout << "done." << endl;
            } else {
                firstConstellation(w,prod);
            }
            deriveWord(w,prod,v);
            list.addEnd(v);
        }
    }

    return;
}

TStrList deriveNextList(TStrList oldList)
{
    TStrList newList;

    while (!oldList.emptyList()) {
        deriveWordList(oldList.data(1),newList);
        oldList.deletEl(1);
    }

    return newList;
}

void derive()
{
    ofstream out(outfile);
    TStrList list;

    if (!out) {
        error(3);
        exit(0);
    }
    list.addEnd(lsystem.omega);
    printList(out,list,0);
    for (unsigned int k = 1; k < step + 1; k++) {
        cout << "step " << k << " :" << endl;
        list = deriveNextList(list);
        printList(out,list,k);
    }
```

```
}

////////////////////// derive ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

int main(int argc, char *argv[])
{
    readinput(argc,argv);

    cout << "infile  = " << infile << "\n";
    cout << "outfile = " << outfile << "\n";
    cout << "limits > 0:\n";
    for (unsigned short int i = 0; i < 256; i++) {
        if (lsystem.limit[i] > 0) {
            cout << "k(" << (char) i << ")    = " << lsystem.limit[i] << "\n";
        }
    }
    cout << "omega   = " << lsystem.omega << "\n";
    cout << "step    = " << step << "\n";
    cout << "productions:\n";
    for (unsigned short int i = 0; i < 256; i++) {
        Table h = lsystem.table[i];
        for (unsigned short int j = 1; j < h.number() + 1; j++) {
            cout << char(i) << "->" << h.data(j) << endl;
        }
    }
    if (TERMINAL)
        cout << "Printing terminal words only.\n";
    else
        cout << "Printing terminal and nonterminal words.\n";

    derive();
}
```

A.4 Simulator for kl0L systems

```
/*
    Author: Markus Seemann
    Date:   2007-07-07
*/
#include <iostream.h>
#include <fstream.h>
#include <string.h>
#include <stdlib.h>
#include <syslimits.h>

#include "DListTemp.h"
#include "mathe.h"

const char LF = '\n';
const char CR = '\r';
const unsigned short int  maxWordLen = USHRT_MAX;
```

```
const unsigned int  maxProdWordLen = 50;

typedef char*               TFileName;
typedef char                TWord[maxWordLen];
typedef char                TProdWord[maxProdWordLen];
typedef unsigned int        TLimit;
typedef stringList          Table;
typedef stringList          TStrList;

class LSystem {
    public:

    TLimit limit;
    char*  omega;
    Table  table[256];

    LSystem(TLimit k = 0, char * s = "")
    {
        limit = k;
        omega = s;
    }
};

unsigned int TERMINAL = 0;    // 0 iff all generated words shall be printed.
unsigned int ALLDERIV = 0;    // 0 iff only distinct words shall be generated.
unsigned int RANDOM   = 0;    // 0 iff all words shall be generated randomly.

TFileName    infile = "", outfile = "outfile.wrd";
unsigned int step  = 1;       // number of derivation steps.
TWord        buffer = "";
LSystem      lsystem(0,buffer);



void copy(char *s, char *t)
// Copies strlen(s)+1 (incl. final '\0') characters from s to t.
{
    for (unsigned int i = 0; i < strlen(s)+1; i++)
        t[i] = s[i];
}

static void usage(const char *progname)
// Print a message describing program and options
{
    cerr << progname << " prints all words generated by a klET0L-system in n derivation steps.\n"
         << "Usage: " << progname <<
            " [-h] [-terminal] [-all] [-random] [-i file] [-o file] [-k limit] [-omega axiom]" <<
            " [-p A P] [-step n]\n"
 << "-h           prints this page\n"
 << "-terminal    only words without symbols A,..,Z are written to output file\n"
 << "-all         all derivable words are generated\n"
 << "-random      from all derivable words in a step one is selected randomly\n"
 << "-i           specifies input file, e.g. (in arbitrary order):\n"
 << "                    k = 5;\n"
```

```
 << "                omega = hello;\n"
 << "                    p = A->cAb;\n"
 << "                     ...\n"
 << "                    p = b->c;\n"
 << "-o           specifies output file (default '" << outfile << "')\n"
 << "-k           specifies the limit of the system (default k = 0)\n"
 << "-omega       specifies the axiom of the system\n"
 << "-p           specifies a production A P of the system\n"
 << "             where A is a symbol and P is a sequence of symbols\n"
 << "-step        number of derivation steps (default n = 1).\n";
}

void error(const int n)
{
    cerr << "***Error(" << n << "): ";
    switch (n) {
case  0: cerr << "No axiom specified!\n";
         break;
case  1: cerr << "Can't open file " << infile << "\n";
         break;
case  2: cerr << "Syntax error in file " << infile << "\n";
         break;
case  3: cerr << "Can't open file " << outfile << "\n";
         break;
case  4: cerr << "Writing error at file " << outfile << "\n";
         break;
case  5: cerr << "Syntax error in parameter specification:\n";
         break;
case  6: cerr << "Wrong number selected.\n";
         break;
default: break;
    }
}

////////////////////// readinput vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

int Getline(ifstream &in, char* str, int size, const char sep = LF)
// Like getline, but ignores white spaces. Returns 0 iff not ok.
{
    while (1) {
        char c = in.peek();
        if ((c == ' ') || (c == LF) || (c == CR) || (c == '\t')) {
            in.get(str[0]);
        } else {
            break;
        }
    }
    return(!(in.getline(str,size,sep)).eof());
}

int isTerminal(char *s)
// Returns 0 iff string s contains a character A,..,Z.
{
    for (unsigned int i = 0; i < strlen(s); i++) {
```

```
        if ( (s[i] >= 'A') && (s[i] <= 'Z') )
            return 0;
    }
    return 1;
}

int addProduction(char c, char* s)
// Returns 0 iff not ok.
{
    if (isTerminal(s)) {
        return lsystem.table[c].addBeg(s);
    } else {
        return lsystem.table[c].addEnd(s);
    }
}

int defParam(ifstream &in, char* str, int size)
// Returns 0 iff no correct definition found.
{
    const char*  keyword[] = {"k","omega","p"};
    const int    keyno = 3;
    TProdWord    pw;

    for (int i = 0; i < keyno; i++) {
        if (!strncmp(str,keyword[i],strlen(keyword[i]))) {   // keyword[i] recognized
            switch (i) {
case  0: // k
         if ( (in >> lsystem.limit) &&
              Getline(in,str,size,';') &&
              (strlen(str) == 0) ) {
     return 1;
 }
 break;
case  1: // omega
         if ( Getline(in,buffer,sizeof(buffer),';') &&
              (strlen(buffer) > 0) ) {
             lsystem.omega = buffer;
     return 1;
 }
          break;
case  2: // p
         if ( Getline(in,str,sizeof(str),'>') &&
              Getline(in,pw,sizeof(pw),';') ) {
             addProduction(str[0],pw);
     return 1;
           }
          break;
default: break;
    }
            error(2);
            return 0;
        }
    }
    error(2);
```

```
    return 0;
}

int parse()
/* Accepted syntax:  (in arbitrary order)
      k = 5;
  omega = hello;
      p = A->cAb;
       ...
      p = b->c;
   Returns 0 iff not ok.
*/
{
    ifstream in(infile);
    char     str[128];

    if (!in) {
        error(1);
        return 0;
    }
    while (Getline(in,str,sizeof(str),'=')) {
        if (!defParam(in,str,sizeof(str))) {
            return 0;
        }
    }
    if (strlen(str) > 0) {
        error(2);
        return 0;
    } else {
        return 1;
    }
}

void readinput(int argc, char *argv[])
// Reads input from command line.
{
    for (int i = 1; i < argc; i++) {
if (!strcmp(argv[i], "-h")) {
         usage(argv[0]);
         exit(0);
} else if (!strcmp(argv[i], "-terminal")) {
    TERMINAL = 1;
} else if (!strcmp(argv[i], "-all")) {
    ALLDERIV = 1;
} else if (!strcmp(argv[i], "-random")) {
    RANDOM = 1;
} else if (!strcmp(argv[i], "-i")) {
    infile = argv[++i];
    if (!parse())
        exit(0);
} else if (!strcmp(argv[i], "-o")) {
    outfile = argv[++i];
} else if (!strcmp(argv[i], "-k")) {
    lsystem.limit = atoi(argv[++i]);
```

```
} else if (!strcmp(argv[i], "-omega")) {
    lsystem.omega = argv[++i];
} else if (!strcmp(argv[i], "-p")) {
    if (i + 1 < argc) {
        addProduction(argv[i + 1][0],argv[i + 2]);
        i += 2;
    } else {
error(5);
usage(argv[0]);
exit(0);
    }
} else if (!strcmp(argv[i], "-step")) {
    step = atoi(argv[++i]);
} else {
         usage(argv[0]);
         exit(0);
  }
    }
    if (strlen(lsystem.omega) == 0){
     error(0);
     usage(argv[0]);
     exit(0);
    }
}

////////////////////// readinput ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

////////////////////// printList vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

void printWord(ofstream &out, char* w, unsigned int s, long int  c)
{
    out << "word" << s << '_' << c << " = " << w << ';' << endl;
    return;
}

void printList(ofstream &out, TStrList l, unsigned int s)
{
    unsigned short int  c = 0;

    if (TERMINAL) {
        for (unsigned short int i = 1; i < l.number() + 1; i++)
            if ( isTerminal(l.data(i)) ) {
                c++;
            }
    } else {
        c = l.number();
    }

//     out << "All " << c << " ";
//     if (TERMINAL) {
//         out << "terminal ";
//     }
//     out << "words generated by " << s << " derivation steps:" << endl;
```

```
    c = 0;
    if (TERMINAL) {
        for (unsigned short int i = 1; i < l.number() + 1; i++)
            if ( isTerminal(l.data(i)) ) {
                c++;
                printWord(out,l.data(i),s,c);
                out << endl;
            }
    } else {
        for (unsigned short int i = 1; i < l.number() + 1; i++) {
            printWord(out,l.data(i),s,i);
            out << endl;
        }
    }
    if (!out) {
        error(3);
        exit(0);
    }

    return;
}

////////////////////// printList ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

////////////////////// derive vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

void deriveWord(char *w, unsigned int prod[], char *v)
// Writes the derivation of w according to prod into the string v.
{
    v[0] = '\0';
    for (unsigned int i = 0; i < strlen(w); i++) {
        if (prod[i] > 0) {
            copy(lsystem.table[w[i]].data(prod[i]), v + strlen(v));
        } else {
            v[strlen(v) + 1] = '\0';
            v[strlen(v)] = w[i];
        }
    }

    return;
}

////////////////////// firstConstellation vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

void firstConstellation(char *w, unsigned int prod[])
{
    unsigned int limitLeft[256];
    // maximum number of occurences in w of every symbol to be rewritten.

    for (int i = 0; i < 256; i++) { limitLeft[i] = lsystem.limit; }

    for (unsigned int i = 0; i < strlen(w); i++) {
        if (limitLeft[w[i]] > 0) {
            limitLeft[w[i]]--;
```

```
            prod[i] = lsystem.table[w[i]].number();  // also works if one symbol has 0 productions
        }
    }

    return;
}

////////////////////// firstConstellation ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

////////////////////// nextConstellation vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

int nextSymbolConf(char *w, unsigned int prod[], char c)
// Returns 0 iff there is no further configuration.
{
    unsigned int pos[maxWordLen]; // positions of the occurrences of c in w.
    unsigned int len = 0;
    unsigned int over = 0;

    for (unsigned int i = 0; i < strlen(w); i++) {
        if (w[i] == c) {
            pos[len] = i;
            len++;
        }
    }

    for (int i = len - 1; i >= 0; i--) {
        if (prod[pos[i]] > 0) {
            over++;
        } else {
            for (int j = i - 1; j >= 0; j--) {
                if (prod[pos[j]] > 0) {        // one further configuration found
                    for (unsigned int k = 0; k < over; k++) {
                        prod[pos[len - k - 1]] = 0;
                    }
                    for (unsigned int k = 0; k < over + 1; k++) {
                        prod[pos[j + k + 1]] = lsystem.table[c].number();
                    }
                    prod[pos[j]] = 0;
                    return 1;
                }
            }
            // no further configuration
            for (unsigned int k = 0; k < over; k++) {
                prod[pos[len - k - 1]] = 0;
            }
            for (unsigned int k = 0; k < over; k++) {
                prod[pos[k]] = lsystem.table[c].number();
            }
            return 0;
        }
    }

    return 0;
}
```

```
int nextConf(char *w, unsigned int prod[], unsigned int ix, char pres[], unsigned int &len)
// Returns 0 iff there is no further configuration, i.e. the choice of the
// symbols to be rewritten.
// ix = int(c) where c is the considered limited symbol.
{
    if (nextSymbolConf(w,prod,pres[ix])) {
        return 1;
    }
    if (ix + 1 < len) {
        return nextConf(w,prod,ix + 1,pres,len);
    }
    return 0;
}

int nextConstellation(char *w, unsigned int prod[], char pres[], unsigned int &len)
// Returns 0 iff there is no further constellation, i.e. the choice of the
// symbols to be rewritten and of the productions.
{
    unsigned int m = strlen(w);

    for (unsigned int i = 0; i < m; i++) {
        if (prod[i] > 1) {
            prod[i]--;
            return 1;
        } else if (prod[i] == 1) {  // overflow -> search next counter
            prod[i] = lsystem.table[w[i]].number();
        }
    }

    // overflow of all counters -> search next configuration
    if (nextConf(w,prod,0,pres,len)) {
        return 1;
    }

    return 0;  // no further derivation found
}

//////////////////// nextConstellation ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

//////////////////// randomConstellation vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

void randomConstellation(
char *w,  // word to rewrite.
unsigned int prod[],  // selected production to be applied for each occurrence within w.
        char pres[],  // string of symbols occurring within w.
        unsigned int &len // length of pres.
        )
// Randomly chooses a constellation.
{
    char a;
    unsigned int wlen = strlen(w);
    unsigned int ix[maxWordLen]; // indexes of remaining occurrences of a within w.
    unsigned int ixlast;  // index in vector ix of the last of such above indexes.
```

```
    unsigned int K;   // number of a's to rewrite.
    unsigned int sel;
    // position of the selected occurrence of a in the vector of unselected occurrences.

    for (unsigned int i = 0; i < len; i++) { // for each cell type a ...
        a = pres[i];
        if (lsystem.table[a].number() > 0) {
            ixlast = 0;
            for (unsigned int j = 0; j < wlen; j++) {   // initialize ix.
                if (w[j] == a) {
                    ix[ixlast] = j; // store index of occurrence a within w.
                    ixlast++;
                }
            }
            K = min(ixlast,lsystem.limit);   // number of a's to rewrite.
            ixlast--; // pointing at the last index of occurrence a within w.
            for (unsigned int j = 0; j < K; j++) {      // select K times.
                sel = random(ixlast); // select occurrence to rewrite.
                prod[ix[sel]] = random(lsystem.table[a].number() - 1) + 1;
                // select production to apply.
                ix[sel] = ix[ixlast];                   // remove selected index from ix.
                ixlast--;
            } // for j
        } // if
    } // for i

    return;
}

////////////////////// randomConstellation ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Integer derivNum(unsigned int parikh[], char pres[], unsigned int &len)
// Returns the number of distinct derivations.
{
    Integer r = 1;

    for (unsigned int i = 0; i < len; i++) {
        char a = pres[i];
        unsigned long int num = lsystem.table[a].number();
        if (num > 0) {
            unsigned int m = min( lsystem.limit, parikh[a] );
            r *= choose(parikh[a],m) * pow(num,m);
        }
    }

    return r;
}

void deriveWordList(char *w, TStrList &list)
{
    unsigned int prod[maxWordLen]; // selected production to be applied for each occurrence within w.
    unsigned int parikh[256];      // parikh vector of w.
    char         pres[256];        // string of symbols occurring within w.
    unsigned int len = 0;       // length of pres.
```

```
    TWord v;
    Integer      num;
    unsigned int sel;

    setRandom(sel);

    if (strlen(w) == 0) {
        list.addEnd(w);
    } else {
        for (unsigned int i = 0; i < maxWordLen; i++) { prod[i] = 0; }
        for (unsigned int i = 0; i < 256; i++) { parikh[i] = 0; }
        for (unsigned int i = 0; i < strlen(w); i++) { parikh[w[i]]++; }
        for (unsigned int i = 0; i < 256; i++) {
            if (parikh[i] > 0) {
                pres[len] = i;
                len++;
            }
        }

        if (ALLDERIV) {
            firstConstellation(w,prod);
            deriveWord(w,prod,v);
            list.addEnd(v);
            while (nextConstellation(w,prod,pres,len)) {
                deriveWord(w,prod,v);
                list.addEnd(v);
            }
        } else if (RANDOM) {
            cout << "Selecting one out of all derivations ... ";
            randomConstellation(w,prod,pres,len);
            cout << "done." << endl;
            deriveWord(w,prod,v);
            list.addEnd(v);
        } else {
            if ( (num = derivNum(parikh,pres,len)) > 1) {
                cout << "Enter an arbitrary number [0.." << LONG_MAX << "] : ";
                cin >> sel;
                setRandom(sel);
                cout << "Selecting one out of " << num << " derivations ... ";
                randomConstellation(w,prod,pres,len);
                cout << "done." << endl;
            } else {
                firstConstellation(w,prod);
            }
            deriveWord(w,prod,v);
            list.addEnd(v);
        }
    }

    return;
}

TStrList deriveNextList(TStrList oldList)
{
```

```
    TStrList newList;

    while (!oldList.emptyList()) {
        deriveWordList(oldList.data(1),newList);
        oldList.deletEl(1);
    }

    return newList;
}

void derive()
{
    ofstream out(outfile);
    TStrList list;

    if (!out) {
        error(3);
        exit(0);
    }
    list.addEnd(lsystem.omega);
    printList(out,list,0);
    for (unsigned int k = 1; k < step + 1; k++) {
        cout << "step " << k << " :" << endl;
        list = deriveNextList(list);
        printList(out,list,k);
    }

}

////////////////////// derive ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

int main(int argc, char *argv[])
{
    readinput(argc,argv);

    cout << "infile  = " << infile << "\n";
    cout << "outfile = " << outfile << "\n";
    cout << "k       = " << lsystem.limit << "\n";
    cout << "omega   = " << lsystem.omega << "\n";
    cout << "step    = " << step << "\n";
    cout << "productions:\n";
    for (unsigned short int i = 0; i < 256; i++) {
        Table h = lsystem.table[i];
        for (unsigned short int j = 1; j < h.number() + 1; j++) {
            cout << char(i) << "->" << h.data(j) << endl;
        }
    }
    if (TERMINAL)
        cout << "Printing terminal words only.\n";
    else
        cout << "Printing terminal and nonterminal words.\n";

    derive();
}
```

A.5 Simulator for uniformly kl0L systems

```
/*
    Author: Markus Seemann
    Date:   2007-07-07
*/
#include <iostream.h>
#include <fstream.h>
#include <string.h>
#include <stdlib.h>
#include <syslimits.h>

#include "DListTemp.h"
#include "mathe.h"

const char LF = '\n';
const char CR = '\r';
const unsigned short int  maxWordLen = USHRT_MAX;
const unsigned int        maxProdWordLen = 50;
const unsigned short int  maxTableNum = 256;

const unsigned short int  maxresChar = 50;
const unsigned short int  maxresK = 1000;
Integer res[maxresChar][maxresK];

typedef char*               TFileName;
typedef char                TWord[maxWordLen];
typedef char                TProdWord[maxProdWordLen];
typedef unsigned int        TLimit;
typedef stringList          Table;
typedef stringList          TStrList;

class LSystem {
    public:

    TLimit limit;
    char*  omega;
    Table  table[maxTableNum];

    LSystem(TLimit k = 0, char * s = "")
    {
        limit = k;
        omega = s;
    }
};

unsigned int TERMINAL = 0;    // 0 iff all generated words shall be printed.
unsigned int ALLDERIV = 0;    // 0 iff only distinct words shall be generated.
unsigned int RANDOM   = 0;    // 0 iff all words shall be generated randomly.

TFileName    infile = "", outfile = "outfile.wrd";
unsigned int step  = 1;       // number of derivation steps.
TWord        buffer = "";
```

```
LSystem      lsystem(1,buffer);



void copy(char *s, char *t)
// Copies strlen(s)+1 (incl. final '\0') characters from s to t.
{
    for (unsigned int i = 0; i < strlen(s)+1; i++)
        t[i] = s[i];
}

static void usage(const char *progname)
// Print a message describing program and options
{
    cerr << progname << " prints all words generated by a klET0L-system in n derivation steps.\n"
         << "Usage: " << progname <<
            " [-h] [-terminal] [-all] [-random] [-i file] [-o file] [-k limit] [-omega axiom]" <<
            " [-p A P] [-step n]\n"
 << "-h           prints this page\n"
 << "-terminal    only words without symbols A,..,Z are written to output file\n"
 << "-all         all derivable words are generated\n"
 << "-random      from all derivable words in a step one is selected randomly\n"
 << "-i           specifies input file, e.g. (in arbitrary order):\n"
 << "                    k = 5;\n"
 << "                omega = hello;\n"
 << "                    p = A->cAb;\n"
 << "                     ...\n"
 << "                    p = b->c;\n"
 << "-o           specifies output file (default '" << outfile << "')\n"
 << "-k           specifies the limit of the system (default k = 1)\n"
 << "-omega       specifies the axiom of the system\n"
 << "-p           specifies a production A P of the system\n"
 << "             where A is a symbol and P is a sequence of symbols\n"
 << "-step        number of derivation steps (default n = 1).\n";
}

void error(const int n)
{
    cerr << "***Error(" << n << "): ";
    switch (n) {
case  0: cerr << "No axiom specified!\n";
         break;
case  1: cerr << "Can't open file " << infile << "\n";
         break;
case  2: cerr << "Syntax error in file " << infile << "\n";
         break;
case  3: cerr << "Can't open file " << outfile << "\n";
         break;
case  4: cerr << "Writing error at file " << outfile << "\n";
         break;
case  5: cerr << "Syntax error in parameter specification:\n";
         break;
case  6: cerr << "Wrong number selected.\n";
         break;
```

```
case  7: cerr << "Limit larger than " << maxresK
              << ". Can't derive number of choices. Continue ...\n";
         break;
case  8: cerr << "Number of different symbols larger than " << maxresChar
              << ". Can't derive number of choices. Continue ...\n";
         break;
default: break;
    }
}

////////////////////// readinput vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

int Getline(ifstream &in, char* str, int size, const char sep = LF)
// Like getline, but ignores white spaces. Returns 0 iff not ok.
{
    while (1) {
        char c = in.peek();
        if ((c == ' ') || (c == LF) || (c == CR) || (c == '\t')) {
            in.get(str[0]);
        } else {
            break;
        }
    }
    return(!(in.getline(str,size,sep)).eof());
}

int isTerminal(char *s)
// Returns 0 iff string s contains a character A,..,Z.
{
    for (unsigned int i = 0; i < strlen(s); i++) {
        if ( (s[i] >= 'A') && (s[i] <= 'Z') )
            return 0;
    }
    return 1;
}

int addProduction(char c, char* s)
// Returns 0 iff not ok.
{
    if (isTerminal(s)) {
        return lsystem.table[c].addBeg(s);
    } else {
        return lsystem.table[c].addEnd(s);
    }
}

int defParam(ifstream &in, char* str, int size)
// Returns 0 iff no correct definition found.
{
    const char*  keyword[] = {"k","omega","p"};
    const int    keyno = 3;
    TProdWord    pw;

    for (int i = 0; i < keyno; i++) {
```

```
        if (!strncmp(str,keyword[i],strlen(keyword[i]))) {   // keyword[i] recognized
            switch (i) {
case  0: // k
         if ( (in >> lsystem.limit) &&
              Getline(in,str,size,';') &&
              (strlen(str) == 0) ) {
     return 1;
 }
 break;
case  1: // omega
         if ( Getline(in,buffer,sizeof(buffer),';') &&
              (strlen(buffer) > 0) ) {
             lsystem.omega = buffer;
     return 1;
 }
          break;
case  2: // p
         if ( Getline(in,str,sizeof(str),'>') &&
              Getline(in,pw,sizeof(pw),';') ) {
             addProduction(str[0],pw);
     return 1;
           }
          break;
default: break;
    }
            error(2);
            return 0;
        }
    }
    error(2);
    return 0;
}

int parse()
/* Accepted syntax:  (in arbitrary order)
      k = 5;
  omega = hello;
      p = A->cAb;
       ...
      p = b->c;
   Returns 0 iff not ok.
*/
{
    ifstream in(infile);
    char     str[128];

    if (!in) {
        error(1);
        return 0;
    }
    while (Getline(in,str,sizeof(str),'=')) {
        if (!defParam(in,str,sizeof(str))) {
            return 0;
        }
```

```
    }
    if (strlen(str) > 0) {
        error(2);
        return 0;
    } else {
        return 1;
    }
}

void readinput(int argc, char *argv[])
// Reads input from command line.
{
    for (int i = 1; i < argc; i++) {
if (!strcmp(argv[i], "-h")) {
         usage(argv[0]);
         exit(0);
} else if (!strcmp(argv[i], "-terminal")) {
    TERMINAL = 1;
} else if (!strcmp(argv[i], "-all")) {
    ALLDERIV = 1;
} else if (!strcmp(argv[i], "-random")) {
    RANDOM = 1;
} else if (!strcmp(argv[i], "-i")) {
    infile = argv[++i];
    if (!parse())
        exit(0);
} else if (!strcmp(argv[i], "-o")) {
    outfile = argv[++i];
} else if (!strcmp(argv[i], "-k")) {
    lsystem.limit = atoi(argv[++i]);
} else if (!strcmp(argv[i], "-omega")) {
    lsystem.omega = argv[++i];
} else if (!strcmp(argv[i], "-p")) {
    if (i + 1 < argc) {
        addProduction(argv[i + 1][0],argv[i + 2]);
        i += 2;
    } else {
error(5);
usage(argv[0]);
exit(0);
    }
} else if (!strcmp(argv[i], "-step")) {
    step = atoi(argv[++i]);
} else {
         usage(argv[0]);
         exit(0);
  }
    }
    if (strlen(lsystem.omega) == 0){
     error(0);
     usage(argv[0]);
     exit(0);
    }
}
```

```
////////////////////// readinput ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

////////////////////// fillProd vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

void fillProd()
// If there is no production for symbol A then A->A is inserted.
{
    char s[] = "a";

    for (unsigned short int i = 0; i < maxTableNum; i++) {
        if (lsystem.table[i].number() == 0) {
            s[0] = i;
            lsystem.table[i].addEnd(s);
        }
    }
    return;
}

////////////////////// fillProd ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

////////////////////// printList vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

void printWord(ofstream &out, char* w, unsigned int s, long int  c)
{
    out << "word" << s << '_' << c << " = " << w << ';' << endl;
    return;
}

void printList(ofstream &out, TStrList l, unsigned int s)
{
    unsigned short int  c = 0;

    if (TERMINAL) {
        for (unsigned short int i = 1; i < l.number() + 1; i++)
            if ( isTerminal(l.data(i)) ) {
                c++;
            }
    } else {
        c = l.number();
    }

//     out << "All " << c << " ";
//     if (TERMINAL) {
//         out << "terminal ";
//     }
//     out << "words generated by " << s << " derivation steps:" << endl;

    c = 0;
    if (TERMINAL) {
        for (unsigned short int i = 1; i < l.number() + 1; i++)
            if ( isTerminal(l.data(i)) ) {
                c++;
                printWord(out,l.data(i),s,c);
```

```
                out << endl;
            }
    } else {
        for (unsigned short int i = 1; i < l.number() + 1; i++) {
            printWord(out,l.data(i),s,i);
            out << endl;
        }
    }
    if (!out) {
        error(3);
        exit(0);
    }

    return;
}

////////////////////// printList ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

////////////////////// derive vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

void deriveWord(char *w, unsigned int prod[], char *v)
// Writes the derivation of w according to prod into the string v.
{
    v[0] = '\0';
    for (unsigned int i = 0; i < strlen(w); i++) {
        if (prod[i] > 0) {
            copy(lsystem.table[w[i]].data(prod[i]), v + strlen(v));
        } else {
            v[strlen(v) + 1] = '\0';
            v[strlen(v)] = w[i];
        }
    }

    return;
}

////////////////////// firstConstellation vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

void firstConstellation(char *w, unsigned int prod[])
{
    for (unsigned int i = 0; i < min(strlen(w),lsystem.limit); i++) {
        prod[i] = lsystem.table[w[i]].number();
        // requires that every symbol has at least 1 production
    }

    return;
}

////////////////////// firstConstellation ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

////////////////////// nextConstellation vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

int nextConf(char *w, unsigned int prod[])
// Returns 0 iff there is no further configuration, i.e. the choice of the occurrences
```

```
// symbols to be rewritten.
{
    unsigned int len  = strlen(w);
    unsigned int over = 0;

    if (len > 0) {
        for (int i = len - 1; i >= 0; i--) {
            if (prod[i] > 0) {
                over++;
            } else {
                for (int j = i - 1; j >= 0; j--) {
                    if (prod[j] > 0) {        // one further configuration found
                        for (unsigned int k = 0; k < over; k++) {
                            prod[len - k - 1] = 0;
                        }
                        for (unsigned int k = 0; k < over + 1; k++) {
                            prod[j + k + 1] = lsystem.table[w[j + k + 1]].number();
                        }
                        prod[j] = 0;
                        return 1;
                    }
                }
                // no further configuration
                return 0;
            }
        }
    }
    return 0;
}

int nextConstellation(char *w, unsigned int prod[])
// Returns 0 iff there is no further constellation, i.e. the choice of the
// symbols to be rewritten and of the productions.
{
    unsigned int m = strlen(w);

    for (unsigned int i = 0; i < m; i++) {
        if (prod[i] > 1) {
            prod[i]--;
            return 1;
        } else if (prod[i] == 1) {  // overflow -> search next counter
            prod[i] = lsystem.table[w[i]].number();
        }
    }

    // overflow of all counters -> search next configuration
    if (nextConf(w,prod)) {
        return 1;
    }

    return 0;  // no further derivation found
}

////////////////////// nextConstellation ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
////////////////////// randomConstellation vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

void randomConstellation(char *w, unsigned int prod[])
// Randomly chooses a constellation.
{
    unsigned int len = strlen(w);
    unsigned int ix[maxWordLen];  // indizes of remaining occurrences.
    unsigned int ixlast = len - 1; // index of the last of such indizes.
    unsigned int K   = min(len,lsystem.limit);
    unsigned int sel; // position of the selected occurrence in the vector of unselected occurrences.

    for (unsigned int j = 0; j < len; j++) {    // initialize ix.
        ix[j] = j;
    }
    for (unsigned int j = 0; j < K; j++) {      // select K times.
        sel = random(ixlast);
        prod[ix[sel]] = random(lsystem.table[w[ix[sel]]].number() - 1) + 1;
        ix[sel] = ix[ixlast];                   // remove selected index from ix.
        ixlast--;
    } // for j

    return;
}

////////////////////// randomConstellation ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

////////////////////// derivNum vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

Integer prodSum(unsigned int ix, unsigned int A, unsigned int B,
                unsigned int parikh[], char pres[], unsigned int &presLen)
// Returns the number of distinct derivations where the first ix symbols of pres have been considered.
// A = number of occurrences still to rewrite;
// B = maximum number of occurrences of the remaining symbols.
{
    if (ix < presLen) {
        Integer r = 0;
        unsigned int pix = parikh[pres[ix]];  // number of of occurrences of symbol pres[ix] in w.
        unsigned int hix = lsystem.table[pres[ix]].number();  // number of productions for pres[ix].

        B -= pix;

        for (unsigned int k = max((int)A - (int)B, 0); k <= min(pix, A); k++) {
            if (res[ix][A-k] == 0)
                res[ix][A-k] = prodSum(ix + 1, A - k, B, parikh,pres,presLen);
            r += choose(pix,k) * pow(hix,k) * res[ix][A-k];
        }
        return r;
    } else {
        return 1;
    }
}

Integer derivNum(char* w, unsigned int parikh[], char pres[], unsigned int &presLen)
```

```
// Returns the number of distinct derivations.
{
    unsigned int wLen = strlen(w);
    unsigned int K    = min(lsystem.limit,strlen(w)); // total number of occurrences to rewrite
    unsigned int B    = wLen;         // maximum number of occurrences of the remaining symbols

    if (lsystem.limit >= maxresK) {
        error(7);
        return 2;
    }
    if (presLen >= maxresChar) {
        error(8);
        return 2;
    }
    for (int i = 0; i < maxresChar; i++)
        for (int j = 0; j < maxresK; j++)
            res[i][j]=0;
    return prodSum(0,K,B,parikh,pres,presLen);
}

////////////////////// derivNum ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

void deriveWordList(char *w, TStrList &list)
{
    unsigned int prod[maxWordLen]; // number of the production to be applied to a symbol of w.
    unsigned int parikh[256];      // parikh vector of w.
    char         pres[256];        // all symbols c with parikh[c] > 0.
    unsigned int len = 0;   // Length of pres.
    TWord        v;
    Integer      num;
    unsigned int sel;

    setRandom(sel);

    if (strlen(w) == 0) {
        list.addEnd(w);
    } else {
        for (unsigned int i = 0; i < maxWordLen; i++) { prod[i] = 0; }
        for (unsigned int i = 0; i < 256; i++) { parikh[i] = 0; }
        for (unsigned int i = 0; i < strlen(w); i++) { parikh[w[i]]++; }
        for (unsigned int i = 0; i < 256; i++) {
            if (parikh[i] > 0) {
                pres[len] = i;
                len++;
            }
        }

        if (ALLDERIV) {
            firstConstellation(w,prod);
            deriveWord(w,prod,v);
            list.addEnd(v);
            while (nextConstellation(w,prod)) {
                deriveWord(w,prod,v);
                list.addEnd(v);
```

```
            }
        } else if (RANDOM) {
            cout << "Selecting one out of all derivations ... ";
            randomConstellation(w,prod);
            cout << "done." << endl;
            deriveWord(w,prod,v);
            list.addEnd(v);
        } else {
            if ( (num = derivNum(w,parikh,pres,len)) > 1) {
                cout << "Enter an arbitrary number [0.." << LONG_MAX << "] : ";
                cin >> sel;
                setRandom(sel);
                cout << "Selecting one out of " << num << " derivations ... ";
                randomConstellation(w,prod);
                cout << "done." << endl;
            } else {
                firstConstellation(w,prod);
            }
            deriveWord(w,prod,v);
            list.addEnd(v);
        }
    }

    return;
}

TStrList deriveNextList(TStrList oldList)
{
    TStrList newList;

    while (!oldList.emptyList()) {
        deriveWordList(oldList.data(1),newList);
        oldList.deletEl(1);
    }

    return newList;
}

void derive()
{
    ofstream out(outfile);
    TStrList list;

    if (!out) {
        error(3);
        exit(0);
    }
    list.addEnd(lsystem.omega);
    printList(out,list,0);
    for (unsigned int k = 1; k < step + 1; k++) {
        cout << "step " << k << " :" << endl;
        list = deriveNextList(list);
        printList(out,list,k);
    }
```

```
}

////////////////////// derive ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

int main(int argc, char *argv[])
{
    readinput(argc,argv);

    cout << "infile  = " << infile << "\n";
    cout << "outfile = " << outfile << "\n";
    cout << "k       = " << lsystem.limit << "\n";
    cout << "omega   = " << lsystem.omega << "\n";
    cout << "step    = " << step << "\n";
    cout << "productions:\n";
    for (unsigned short int i = 0; i < 256; i++) {
        Table h = lsystem.table[i];
        for (unsigned short int j = 1; j < h.number() + 1; j++) {
            cout << char(i) << "->" << h.data(j) << endl;
        }
    }
    if (TERMINAL)
        cout << "Printing terminal words only.\n";
    else
        cout << "Printing terminal and nonterminal words.\n";

    fillProd();

    derive();
}
```

A.6 Auxiliary Class "DListTemp.h"

```
/*
    Author: Markus Seemann
    Date:   2007-07-07
*/
#include <iostream.h>

template <class T> class DListEl
{
    public:

    T           Data;
    DListEl<T>* Prev;
    DListEl<T>* Next;

    DListEl(T d, DListEl<T>* pr = NULL, DListEl<T>* ne = NULL)
    {
        Data = d;
        Prev = pr;
```

```
        Next = ne;
    };
};

template <class T> class DList
{
    private:

    unsigned long int  Number; // Number of elements of this list.
    DListEl<T> *        firstEl; // First element of this list.

    DListEl<T> * find(unsigned long int n)
    // Searches the n-th element. Returns NULL iff not found.
    {
        DListEl<T> * p = firstEl;

        if ( (Number == 0) || (n > Number) ) {
            return NULL;
        }
        for (unsigned long int i = 1; i < n; i++) {
            p = p->Next;
        }

        return p;
    }

    public:

    DList() // Constructor.
    {
        Number = 0;
        firstEl = NULL;
    }

    unsigned short int emptyList() { return !Number; }
    // Returns 1 if this list is empty, else 0.

    unsigned long int number() { return Number; }

    T data(unsigned long int n)
    // Returns Data of the n-th element if exists, else NULL.
    {
        DListEl<T> * p = find(n);

        return p->Data;
    }

    int insert(T d, unsigned long int n)
    // Inserts d behind the n-th element. Returns 0 iff not ok.
    {
        DListEl<T> * p = find(n);

        if (n > Number) {
            return 0;
```

```
    }
    if (n == 0) {                    // first element
        p = new DListEl<T>(d,NULL,firstEl);
        if (p == NULL)
            return 0;
        firstEl = p;
        if (Number > 0)
            p->Next->Prev = p;
    } else {
        p->Next = new DListEl<T>(d,p,p->Next);
        p = p->Next;
        if (p == NULL)
            return 0;
        if (n < Number)
            p->Next->Prev = p;
    }
    Number++;

    return 1;
}

int addBeg(T d)
// Inserts d as the first element. Returns 0 iff not ok.
{
    return insert(d,0);
}

int addEnd(T d)
// Inserts d as the last element. Returns 0 iff not ok.
{
    return insert(d,Number);
}

int deletEl(unsigned long int n)
// Deletes the n-th element. Returns 0 iff not ok.
{
    DListEl<T> * p = find(n);

    if ( (n == 0) || (n > Number) ) {
        return 0;
    }
    if (n == 1) {
        firstEl = p->Next;
    } else {                         // n > 1
        p->Prev->Next = p->Next;
    }
    if (n < Number) {
        p->Next->Prev = p->Prev;
    }
    delete(p);
    Number--;

    return 1;
}
```

```
    void deleteList()
    // Deletes the whole list.
    {
        while (deletEl(1)) { }
        return;
    }
};

class stringList
{
    typedef char * T;

    private:

    DList<T>    List;

    public:

    inline unsigned short int emptyList() {
        return List.emptyList();
    }

    inline unsigned long int number() {
        return List.number();
    }

    inline T data(unsigned long int n) {
        if ( (n == 0) || (n > number()) )
            return "";
        return List.data(n);
    }

    inline int insert(T d, unsigned long int n) {
        unsigned short int s = strlen(d) + 1;
        T                  h = (T) operator new (s);

        for (unsigned short int i = 0; i < s; i++)
            h[i] = d[i];
        return List.insert(h,n);
    }

    inline int addBeg(T d) {
        return insert(d,0);
    }

    inline int addEnd(T d) {
        return insert(d,number());
    }

    inline int deletEl(unsigned long int n) {
        if ( (n == 0) || (n > number()) )
            return 0;
        T p = data(n);
```

```
        delete(p);
        return List.deletEl(n);
    }

    void deleteList()
    {
        List.deleteList();
        return;
    }
};
```

A.7 Auxiliary Class "mathe.h"

```
/*
    Author: Markus Seemann
    Date:   2007-07-07
*/
#include <stdlib.h>
#include <syslimits.h>
/*
 #include "Integer.h"
*/
#define Integer unsigned long int

#define max(A,B) ((A) > (B) ? (A) : (B))
#define min(A,B) ((A) < (B) ? (A) : (B))

Integer sum(unsigned long int a, unsigned long int b)
{
    Integer r = 0;

    for (unsigned long int i = a; i <= b; i++) {
        r += i;
    }
    return r;
}

Integer prod(unsigned long int a, unsigned long int b)
{
    Integer r = 1;

    for (unsigned long int i = a; i <= b; i++) {
        r *= i;
    }
    return r;
}

Integer choose(unsigned long int n, unsigned long int k)
{
    if (k > n)
        return 0;
```

```
    if (k > n - k)
        k = n - k;

    return (Integer) ( prod(n - k + 1, n) / prod(1, k) );
}

Integer pow(unsigned long int n, unsigned long int k)
{
    Integer r = 1;

    if (k == 0)
        return 1;
    else if (n == 0)
        return 0;

    while (k) {
        while (!(k & 1)) { // first bit unset
            k >>= 1;
            n *= n;
        }
        k >>= 1;
        r *= n;
        n *= n;
    }

    return r;
}

template <class T> T random(T range)
// Returns a random value between 0..range.
{
    return (T)(lrand48() * ((double)(range + 1) / LONG_MAX));
}

void setRandom(unsigned long int n)
{
    srand48(n);
    return;
}
```

Bibliography

[1] H. Abelson, A. A. diSessa: *Turtle Geometry*. M.I.T. Press, Cambridge, 1982.

[2] N. Chomsky: Three Models for the Description of Languages. *IRE Transactions on Information Theory* **2**(1956), 113–124.

[3] N. Chomsky: On Certain Formal Properties of Grammars. *Information and Control* **2**(1959), 137–167.

[4] J. Dassow: A Remark on Limited 0L Systems. *J. Inf. Process. Cybern. EIK* **24**(1988), 287–291.

[5] J. Dassow, G. Păun, A. Salomaa: On the Union of 0L Languages. *Inform. Process. Lett.* **43**(1993), 59–63.

[6] J. Dassow, D. Wätjen: On the Relations between the Degree of Synchronization and the Degree of Nondeterminism in k-limited and Uniformly k-limited T0L Systems. *Intern. J. Computer Math.* **35**(1990), 69–82.

[7] N. Dunford, J.T. Schwartz: *Linear Operators. Part I. General Theory*. John Wiles and Sons, 1988.

[8] H. Fernau: Membership for 1-limited ET0L Languages is Not Decidable. *J. Inf. Process. Cybern. EIK* **30**(1994), 191–211.

[9] H. Fernau: Membership for k-limited ET0L Languages is Not Decidable. *J. Automata, Languages and Combinatorics* **1**(1996), 243–245.

[10] H. Fernau: Remarks on Propagating Partition-Limited ET0L Systems. *J. Universal Computer Science* **2**(1996), 745–755.

[11] M. Frings: Systeme mit eingeschränkter paralleler Ersetzung. Diplomarbeit, TU Braunschweig, 1985.

[12] S. Gärtner: Partitions-limitierte Lindenmayer-Systeme. Dissertation, TU Braunschweig. *Berichte aus der Informatik*. Shaker-Verlag, Aachen 1995.

[13] S. Gärtner: On Partition Limited 0L Systems. *Developments in Language Theory II*. World Scientific, Singapore 1996, 230–236.

[14] M. Gardner: Mathematical Games – in which "monster" curves force redefinition of the word "curve". *Scientific American* **235(6)**(1976), 124–134.

[15] G. T. Herman, G. Rozenberg: *Developmental Systems and Languages*. North-Holland, 1975.

[16] A. Lindenmayer: Mathematical Models for Cellular Interactions in Development, Part I: Filaments with one-sided inputs. *J. Theoretical Biology* **18**(1968), 280–299.

[17] A. Lindenmayer: Mathematical Models for Cellular Interactions in Development, Part II: Simple and branching filaments with two-sided inputs. *J. Theoretical Biology* **18**(1968), 300–315.

[18] A. Lindenmayer: Developmental Systems without Cellular Interactions, their Languages and Grammars. *J. Theoretical Biology* **30**(1971), 455–484.

[19] A. Lindenmayer, P. Prusinkiewicz: *The Algorithmic Beauty of Plants*. Springer-Verlag, New York 1990.

[20] M. Lothaire: *Combinatorics on Words*. Addison-Wesley, London 1983.

[21] B. B. Mandelbrot: *The Fractal Geometry of Nature*. W. H. Freeman, San Francisco 1982.

[22] D. Ostrovsky, D. Wätjen: Function-Limited 0L Systems Revisited. *J. Automata, Languages and Combinatorics* **6**(2001), 97–114.

[23] P. Prusinkiewicz: Score generation with L-systems. *Proceedings of the International Computer Music Conference*, (1986), 455–457.

[24] G. Rozenberg, P.D. Doucet: On 0L-Languages. *Information and Control* **19**(1971), 302–318.

[25] G. Rozenberg, A. Salomaa (Eds.): *L Systems.* Berlin Heidelberg 1974.

[26] G. Rozenberg, A. Salomaa: *The Mathematical Theory of L Systems.* Academic Press, New York 1980.

[27] G. Rozenberg, A. Salomaa: *The Book of L.* Springer-Verlag, Berlin Heidelberg 1986.

[28] G. Rozenberg, A. Salomaa (Eds.): *Lindenmayer Systems. Impacts on Theoretical Computer Science, Computer Graphics, and Developmental Biology.* Springer-Verlag, Berlin Heidelberg 1992.

[29] A. Salomaa: *Formal Languages.* Academic Press, New York 1973.

[30] M. Seemann: Multiple-limited ET0L Systems. *Proceedings of the* 3rd *International Conference "Developments in Language Theory"*, Aristotle University of Thessaloniki (1998), 377–385.

[31] R. Siromoney, G. Siromoney: Parallel 0L-Languages. *Intern. J. Computer Math.* **5**(1975), 109–123.

[32] H. Spilker, D. Wätjen: Some Undecidability Results Concerning k-limited 0L Systems. *Fundamenta Informaticae* **26**(1996), 23–30.

[33] H. Spilker, D. Wätjen: Decidability Results Concerning k-limited ED0L Systems. *Inform. Process. Lett.* **59**(1996), 13–17.

[34] A. L. Szilard, R. E. Quinton: An Interpretation for D0L Systems by Computer Graphics. *The Science Terrapin* **4**(1979), 8–13.

[35] A. Thue: Probleme über die Veränderung von Zeichenreihen nach gegebenen Regeln. *Christiana* (1914).

[36] D. Wätjen: k-limited 0L Systems and Languages. *J. Inf. Process. Cybern. EIK* **24**(1988), 267–285.

[37] D. Wätjen: On Unary k-limited 0L Systems and Languages. *J. Inf. Process. Cybern. EIK* **24**(1988), 415–429.

[38] D. Wätjen: On k-uniformly-limited 0L Systems and Languages. *J. Inf. Process. Cybern. EIK* **26**(1990), 229–238.

[39] D. Wätjen: Restriction of Active Symbols in k-limited ET0L Systems and a Normal Form Theorem. *Intern. J. Computer Math.* **40**(1991), 47–62.

[40] D. Wätjen: A Weak Iteration Theorem for k-limited E0L Systems. *J. Inf. Process. Cybern. EIK* **26**(1992), 37–40.

[41] D. Wätjen: *Theoretische Informatik*. R. Oldenbourg Verlag, Mnchen Wien 1994.

[42] D. Wätjen: k-limited ED0L Languages are Context-Sensitive. *Bull. EATCS* **61**(1997), 89–91.

[43] D. Wätjen: Undecidability Results for Uniformly k-limited 0L Systems. *J. Automata, Languages and Combinatorics* **5**(2000), 159–167.

[44] D. Wätjen, D. Ostrovsky: Function-Limited 0L Systems Revisited. *J. Automata, Languages and Combinatorics* **6**(2001), 97–114.

[45] D. Wätjen, E. Unruh: On the Degree of Synchronisation of klT0L and klET0L Systems. *Inform. Process. Lett.* **29**(1988), 87–89.

[46] D. Wätjen, E. Unruh: On Extended k-uniformly-limited T0L Systems and Languages. *J. Inf. Process. Cybern. EIK* **26**(1990), 238–299.

Index

$\mathcal{L}$(cf), 12
$\mathcal{L}$(cs), 12
$\mathcal{L}$(fin), 12
$\mathcal{L}$(re), 12
$\mathcal{L}$(rec), 12
$\mathcal{L}$(reg), 12
$FS(K)$, 68
$\#_a w$, 7
$\Longrightarrow$, 11
$\Longrightarrow_G$, *see* $\Longrightarrow$
ε, 8
$\liminf_{n\to\infty}$, *see* limit inferior
$\limsup_{n\to\infty}$, *see* limit superior
L^+, 8
L^*, 8
$\mathbb{N}$, 7
$\mathbb{N}_0$, 7
$\wp(S)$, 7
w^i, L^i, 8
$\mathbb{Z}$, 7
0L system
    limited
        k-limited, 20
        multi-limited, 23
        multi-partition-limited, 105
        partition-limited, 4
        uniformly k-limited, 3
    non-limited, 15
0S system, 51

abstract family of languages, *see* AFL
AFL, 13
    anti-, 14
    full, 14
alphabet, 7
anti-AFL, 14

branching structure, 34

Chomsky Hierarchy, 12
concatenation
    of languages, 8
    of words, 8
cyclic occurrence, 62

derivation path, 27

ET0L system, 15

FASS curve, 31
formal language, *see* language

grammar, 11
    of type i, 11

hexagonal Gosper curve, 31
homomorphism, 9
    ε-free, 9
    inverse, 9

k-limitation, 20
(κ, P)-limitation, 105
κ-limitation, 23

klET0L system, 20
Koch curve, 32

language, 8
    concatenation, 8
    ε-free, 8
    ε-free iteration, 9
    finite, 8
    i-th power, 8
    iteration, 9
letter, 7
limit inferior, 91
limit superior, 91
Lindenmayer system, *see* 0L system

mirror image, 55
mlET0L system, 23
    normal form, 57
monoid, 8
    free, 9
    homomorphism, 9
mplET0L system, 105

normal form of mlET0L systems, 57

prefix, 7
production, 11
productive occurrence, 28

rewriting system, 11

sub-sequence, 92
substitution, 9
    ε-free, 9
    finite, 9
    non-empty, 9
subword, 7
suffix, 7

turtle, 30
    bracketed, 34
    interpretation, 31
type
    cube
      of 0L languages, 19
      of ml0L languages, 26
      of mlT0L languages, 45
    of 0L systems and languages, 15
    of kl0L systems and languages, 20
    of ml0L systems and languages, 24

weak iteration theorem, 61
word, 7
    concatenation, 8
    empty, 8
    i-th power, 8

yield relation, *see* $\Longrightarrow$